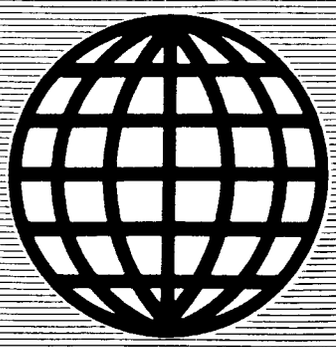


The COMPUTER JOURNAL®

Programming - User Support

Applications

Issue Number 28

$3.00

Starting Your Own BBS

What it Takes to Run a BBS

Build an A/D Converter for the Ampro L.B.

A Simple Low-Cost A/D Interface

The Hitachi HD64180

New Life for 8-bit Systems — Part 2

Using SCSI for Real Time Control

Separating the Memory and I/O Buses

An Open Letter to STD-Bus Manufacturers

Getting an Industrial Control Job Done

Programming Style

User Interfacing and Interaction

Choosing a Language for Machine Control

Using HTPL as a CNC Language

Patching Turbo Pascal

Removing the blasted ":=" Requirement

ISSN # 0748-9331

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, Montana
59912
406-257-9119

Editor/Publisher
Art Carlson

Art Director
Donna Carlson

Production Assistant
Judie Overbeek

Circulation
Donna Carlson

Contributing Editors
C.Thomas Hilton
Donald Howes
Bill Kibler
Rick Lehrbaum
Peter Ruber
Jay Sage
Jon Schneider

Entire contents copyright© 1987 by The Computer Journal.

Subscription rates—$16 one year (6 issues), or $28 two years (12 issues) in the U.S., $22 one year in Canada and Mexico, and $24 (surface) for one year in other countries. All funds must be in US dollars on a US bank.

Send subscriptions, renewals, or address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, Montana, 59912, or The Computer Journal, PO Box 1697, Kalispell, MT 59903.

Bulletin Board—Our bulletin board will be on line 24 hours a day at 300 and 1200 baud, and the number is (406) 752-1038.

Address all editorial and advertising inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912 phone (406) 257-9119.

The COMPUTER JOURNAL

Features

Starting Your Own BBS
Running a board takes more than equipment, it also takes time and skils — by R.E. McCain 6

Buiild an A/D Converter for the Ampro L.B.
A one-chip A/D converter for digitizing and isolating noisy low frequency signals — by John B. Rasor D.O. 9

The Hitachi HD64180
Setting the HD64180's wait states and RAM refresh rates, using the Programmable Reload Timers, and the Direct Memory Access capabilities — by Jon Schneider 12

Using SCSI for Real Time Control
Early buses supported both memory and I/O expansion within a single set of signals, and suffered from the limitations of the required compromises — by Rick Lehrbaum .. 25

An Open Letter to STD-Bus Manufacturers
Comments on what is needed to make machines and research labs go — by Jerry Nelson, Ph.D. 29

Programming Style
Design of the user/program interaction is at least as important as the problem solving algorithms by Art Carlson 37

Patching Turbo Pascal
Using disassembled Z-80 source code to modify Turbo Pascal to suit your style — by Clark A. Calkins 39

Choosing a Language for Machine Control
The advantages of a compiled RPN Forth-like language for Computer Numerical Control — by Joe Bartel 41

Columns

Editorial 3

Reader's Feedback 4

Z-SIG Corner
by Jay Sage 17

News 36

Computer Corner
by Bill Kibler 44

Z sets you free!

WHO WE ARE
Echelon is a unique company, oriented exclusively toward your CP/M-compatible computer. Echelon offers top quality software at extremely low prices; our customers are overwhelmed at the amount of software they receive when buying our products. For example, the Z-Com product comes with approximately 80 utility programs; and our TERM III communications package runs to a full megabyte of files. This is real value for your software dollar.

ZCPR3
Echelon is famous for our operating systems products. ZCPR3, our CP/M enhancement, was written by a software professional who wanted to add features normally found in minicomputer and mainframe operating systems to his home computer. He succeeded wonderfully, and ZCPR3 has become the environment of choice for "power" CP/M users.

Multiple Commands per Line
You can easily use multiple commands per line under ZCPR3. Simply separate the individual commands with semicolons. For example, "PIP B:=A:*.TXT;STAT B:*.*" will copy files and then show you the STAT results.

User-Programmed menu systems
ZCPR3 comes with three different menu systems that you can use to create custom menu-driven "front ends" for your computer. This is especially useful for setting up menus for your spouse or co-workers to use the computer, as they never have to see the A> prompt. All they have to do is press a single key to run any single or multiple CP/M programs, and when the task is done, control is automatically returned to the menu (ordinary CP/M menu programs cannot do this).

Extended Command Processing
When you type a command under CP/M, it will only look for the program in the current drive and user area. ZCPR3 gives you more flexibility by additionally searching other disks and user areas when resolving commands. You have full control of this function, called the PATH. This is probably the one element of ZCPR3 that is missed most if you return to "ordinary" CP/M.

Also, ZCPR3 supports the capability of grouping all your commonly used utility programs into a library file (*.LBR). This is great for systems with a small number of directory entries per disk, as the library file only uses one entry. It also has the advantage of reducing disk space requirements for a given set of programs, allowing you to put more programs on a disk. And the programs in the library file are invokable from the command line just like any other program not in the library.

Other Features
There's much more to ZCPR3, like named directories, online help system, etc., but it can't be described on one page. If you would like more information, consider the books shown below.

Z-SYSTEM
Perhaps the only shortcoming of ZCPR3 is that it is not a complete replacement for CP/M. This is what the Z-System does. The Z-System contains ZCPR3 and an additional module, ZRDOS, and is a complete replacement for CP/M. ZRDOS adds even more utility programs, and has the nice feature of no need to warm boot (^C) after changing a disk. Hard disk users can take advantage of ZRDOS "archive" status file handling to make incremental backup fast and easy. Because ZRDOS is written to take full advantage of the Z80, it executes faster than ordinary CP/M and can improve your system's performance by up to 10%.

INSTALLING ZCPR3/Z-SYSTEM
Echelon offers ZCPR3/Z-System in many different forms. For $44 you get the complete source code to ZCPR3 and the installation files. However, this takes some experience with assembly language programming to get running, as you must perform the installation yourself.

For users who are not qualified in assembly language programming, Echelon offers our "auto-install" products. Z-Com is our 100% complete Z-System which even a monkey can install, because it installs itself. Z-Com includes many interesting utility programs, like UNERASE, MENU, VFILER, and much more.

Echelon also offers "bootable" disks for some CP/M computers, which require absolutely no installation, and are capable of reconfiguration to change ZCPR3's memory requirements. At present, only Kaypro computers have this option available.

BOOKS
We sometimes joke around the office that we are really in the business of publishing, not selling software. We have books. Lots of books. We have to have lots of books, considering how powerful our software is and the large quantity of different packages we offer. Here are our best sellers:

ZCPR3: The Manual
This is the "bible" for the ZCPR3 user. An exhaustive technical reference, bound softcover, 350 pages. Contains descriptions of each ZCPR3 utility program, a detailed discussion about the innards of ZCPR3, and a full installation manual for those doing their own installation. You could order it from B. Dalton, but why? Get it from us.

The Z-System User's Guide
For those who are not technically inclined. This is an excellent tutorial-style manual filled with examples of how to use the power of ZCPR3/Z-System most effectively, written by two highly experienced Z users. (One user is a lawyer, the other a writer; this proves that anyone can use Z and benefit from it.)

ZCPR3: The Libraries
The extensive documentation for the libraries of ZCPR3, SYSLIB, Z3LIB, and VLIB. A must for any serious user of these programming tools. Loose-leaf notebook style; easy to work with as it will lay flat on your desk.

THERE'S MORE
We couldn't fit all Echelon has to offer on a single page (you see how small this type is). We haven't begun to talk about the many additional software packages and publications we offer. Send in the order form below and just check the "Requesting Literature" box for more information.

Item	Name	Price
1	ZCPR3 Core Installation Package	$44.00 (3 disks)*
2	ZCPR3 Utilities Package	$89.00 (9 disks)
3	Z3-Dot-Com (Auto-Install ZCPR3)	$99.00 (6 disks)*
4	Z3-Dot-Com "Bare Minimum"	$49.95 (1 disk)
5	Z-Com (Auto-Install Z-System)	$119.00 (7 disks)*
6	Z-Com "Bare Minimum"	$69.95 (2 disks)
12	PUBLIC ZRDOS Plus (by itself)	$59.50 (1 disk)
13	Kaypro Z-System Bootable Disk	$69.95 (3 disks)
20	ZAS/ZLINK Macro Assembler and Linker	$69.00 (1 disk)
21	ZDM Debugger for 8080/Z80/HD64180 CPU's	$50.00 (1 disk)
22	Translators for Assembler Source Code	$51.00 (1 disk)
23	REVAS3/4 Disassembler	$90.00 (1 disk)
24	**Special** – Items 20 through 23	$150.00 (4 disks)
25	DSD-80 Full Screen Debugger	$129.95 (1 disk)
27	The Libraries. SYSLIB, Z3LIB, and VLIB	$69.00 (8 disks)
28	Graphics and Windows Libraries	$49.00 (1 disk)
29	**Special** – Items 27, 28, and 82	$129.00 (9 disks)
40	Input/Output Recorder IOP (I/OR)	$39.95 (1 disk)
41	Background Printer IOP (BPrinter)	$39.95 (1 disk)
42	Programmable Key IOP (PKey)	$39.95 (1 disk)
43	**Special** – Items 40 through 42	$89.95 (3 disks)
60	DISCAT Disk cataloging system	$39.99 (1 disk)
61	TERM3 Communications System	$99.00 (6 disks)
64	Z-Msg Message Handling System	$99.00 (1 disk)
81	**ZCPR3: The Manual** bound, 350 pages	$19.95
82	**ZCPR3: The Libraries** 310 pages	$29.95
83	Z-NEWS Newsletter, 1 yr subscription	$24.00
84	**ZCPR3 and IOPs** 50 pages	$9.95
85	**ZRDOS Programmers's Manual** 35 pages	$8.95
88	**Z-System User's Guide** 80 page tutorial	$14.95

*Includes **ZCPR3: The Manual**

Echelon, Inc.
885 N. San Antonio Road, Los Altos, CA 94022 USA
415/948-3820 (order line and tech support)

NAME _____
ADDRESS _____
_____
TELEPHONE _____ DISK FORMAT _____
☐ REQUESTING LITERATURE

ORDER FORM

Payment to be made by:
☐ Cash
☐ Check
☐ Money Order
☐ UPS COD
☐ Mastercard/Visa:
_____
Exp. Date _____

California residents add 7% sales tax.
Add $4.00 shipping/handling.

ITEM	PRICE
_____	_____
_____	_____
_____	_____
Subtotal	_____
Sales Tax	_____
Shipping/Handling	_____
Total	_____

Editor's Page

Which Comes First The Chicken or the Egg?

I am basically an experimenter who wants to use computers to perform tasks, and most of the things which I want to do are unconventional—I am basically not a programmer or a hardware designer, but I have to work in both of these areas in order to accomplish my goals. I have no desire to write large elegant programs or to design new computer systems, but I can write small to moderate size programs or modify something which almost works if I have the source code and I can wirewrap simple boards or do a hardware patch on an existing board. But, I get very frustrated because very few things do what I need to do the way I want them done!

I've come to the conclusion that too many people are taking an existing operating system or hardware system and then trying to force it to do everything. I contend that this is backwards! We should first determine what it is that we want to do, and then determine what we need in order to do it. Both hardware and software are cheap compared with what they were a few years ago, and we should select the tools needed for the individual job.

I've spent too much time trying to find what I need, but all I've found are elaborate systems which attempt to do everything for everyone with fancy graphics and the designer's idea of a user's interface—the problem is that they don't do the simple things that I need the way that they should be done.

For me, it's back to the basics (that's basics as in fundamentals, not BASIC the language), and I'm not going to be concerned about being compatible with anyone else or using the hottest new devices. I'll just use whatever enables me to get the particular job done whether it's my old Apple II+ or one of the 32-bit chips with a coprocessor. If need be, I'll start with a bare board and wirewrap (groan).

Ten or fifteen years ago what we could accomplish was limited by the devices we had to work with, today what we can accomplish is only limited by what we can think of! And I can think of lots of things which are difficult to implement because of the sytems available. The devices exist; but the hardware and software systems are unsatisfactory, and I intend 'roll my own.'

My approach is to analyse the application to determine the requirements for the particular job, and then to compare these with the capabilities of what is available. If something does the job well, I'll use it—if something almost does the job I'll consider modifying it—and if I can't find anything which does the job the way I want it done, I'll design something which does. The guiding principle for industrial and control applications is KISS (Keep It Simple Stupid), and my guidelines for judgement include doing it well without unnecessary embelishments because these additional unused features result in slower response, larger size, higher cost, and/or poorer reliability.

"....today what we can accomplish is only limited by what we can think of!"

This is different than the usual approach of putting a PC in an office or on a laboratory desk, and then forcing it to do everthing regardless of how poorly it is suited for the job. But I'm talking about an entirely different concept in the machine shop, in the factory, in the chemical plant, or at a remote location, where the normal desk top computer is not suitable, and where the size, reliability, and other requirements combined with the critical nature and/or production volume justify the effort in designing a package for the specific application.

During the next few years, the desk top hardware and software systems will become so complex, and so readily available from major suppliers, that there will be little need for us little guys in the major market. We'll have to work in a niche market such as the industrial and control field which is growing very rapidly, and which can not be so easily dominated by a few major suppliers because of the fact that things have to be tailored for each specific application. Just as microcomputers for individuals replaced the all-purpose group-user mainframes, dedicated microcontrollers optimized for specific applications will replace general purpose microcomputers for many uses.

Achievers vs. Chrome Collectors

Some photographers, will always tell you about the outstanding pictures they are going to take REAL SOON NOW when they can afford the newest, state-of-the-art, high priced equipment. It doesn't matter that they already have a room full of equipment—as soon as something new is announced they set their current equipment aside and wait till they can obtain the latest creation. And of course, by the time they have it, something even more advanced will have been announced and they'll repeat the never ending cycle. These photographers are called 'chrome collectors' because they try to solve their problems by spending money on equipment instead of spending the effort to obtain the maximum potential of what they already have, and all too often some kid with a very limited budget but a lot of smarts will buy an old camera for a few bucks at a rummage sale and produce prize winning pictures while the chrome collector is chasing the non-existant perfect camera.

Most of us serious computerists are guilty of lusting for every new computer and peripherial as soon as it is announced—I freely admit that I'd like to have every system ever made, and the time to do nothing but learn all about every one of them—but we have to face the fact that in the majority of the cases we have not

(Continued on page 32)

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these registered trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used marks are acknowledged, and we apologize for any we have overlooked.

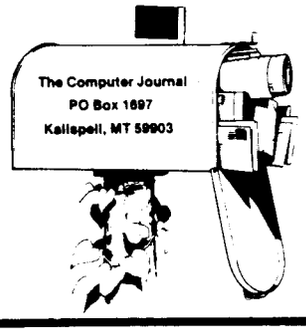
Apple II, II + , IIc, IIe, Macintosh, DOS 3.3, ProDOS; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. MBASIC; Microsoft. Wordstar; MicroPro International Corp. IBM-PC, XT, and AT; IBM Corporation. Z-80, Zilog. MT-BASIC, Softaid, Inc. Turbo Pascal, Borland International.

Where these terms (and others) are used in The Computer Journal they are acknowledged to be the property of the respective companies even if not specifically mentioned in each occurence.

Make certain that TCJ follows you to your new address. Send both old and new address along with your expiration number that appears on your mailing label to:

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, MT 59912

If you move and don't notify us, TCJ is not responsible for copies you miss. Please allow six weeks notice. Thanks.



Reader's

Problems With Smartkey

In issue #24, Robert W. talks about using Smartkey with Wordstar.

My problem is more basic. How do you use smartkey? I just can't get Smartkey to work because I can't, not because it won't.

I am using an Xerox which I assembled from here & there. It is a model 820-II etch 2 with two 8" floppys. It uses ZCPR2 on the system program. I did not do it.

Can anyone help?

Hiram De Santis
1896 Keewin Ave. NE
Palm Bay, FL 32905

HD64180 Fan

Bill Kibler,

I am responding to your Computer Corner column in TCJ #27, the one you wrote on Thanksgiving on your Heath/Zenith 171.

How do I link the MicroMint SB-180 to the STD bus using Ampro's products? Here is some background.

I want target system and development system to both remain 8-bit, and the development system must be based on the Hitachi HD64180. Accept these as givens; the background for those decisions is available for your critical comments on my "open letter to STD Bus manufacturers."

It doesn't HAVE to be a MicroMint SB-180. But do you know of any other HD64180 which comes closer to being available as a turn-key system? With as much support? Price/performance?

It doesn't have to be separate development system and STD bus system, but do you know of any turn-key HD64180 system which itself resides on the STD bus? There are some CPU cards listed on the enclosed open letter, but where are the configured systems? Every manufacturer assumes every developer wants to develop on a PC. Compared to a 4 MHz, 64K vanilla CP/M system, I would prefer the PC too. But with the availability of a faster 6 to 10 MHz Hitachi 512K system, I would prefer the Hitachi, not just for speed, but for simplicity; target and development system are CPU and instruction set identical; no cross software, no in-circuit emulation.

It doesn't have to be an Ampro SCSI/IOP intelligent I/O processor on the STD bus and a SCSI socket on the development system. But do you know of any simpler or more complete or better supported solution to linking a computer and an STD Bus expansion chassis together as one system?

The state of AMPRO: they have available an 80186 system with linked STD bus, complete. But we want 8 bit. The Z80 SBC has no SCSI, but you can unplug the Z80 and plug in a piggy-back in order to add a SCSI port. The STD bus, of course, gets the SCSI/IOP, but AMPRO has no software to integrate the STD expansion with the ZDOS BIOS. As for HD64180, it is not in their product line. You can unplug the Z80 an add a 64180 plus RAM from M.A.N. Systems, but then there goes ability to retrofit a SCSI port. Clearly one needs a 64180 card designed with all the RAM and the SCSI from the ground up. That's where the MicroMint SB-180FX comes in, but who has configured THAT into a complete system with STD cage?

"The driving force is neither software or hardware, but rather the solutions to problems" (Art Carlson, Editor, TCJ).

So there it is. How would you put it together?

Jeremiah I. Nelson, Ph.D.
Working Group in Biophysics
Philipps University; Renthof 7
D-3550 Marburg (tel:06421/284161)

Editor's Note: Ampro's Z80 Little Board Plus now has the SCSI port on the board and SCSI support software is also available, so the M.A.N. 64180 board can be added and still provide the SCSI port.

Feedback

Dr. Nelson's open letter to STD Bus manufactures is reprinted in this issue. Send your feedback to Nelson either directly to him (with a copy to TCJ), or send it to us and we'll forward it.

Response to Dr. Nelson's Letter

Let me say I am happy to get the chance to comment on your letters that bring up some interesting questions. The STD Bus is doing pretty well, but is lacking in some areas. As a fan of Forth I have noticed that not many STD Bus people offer Forth as an operating system, as they once did. I feel your main complaint was the lack of complete systems, turn-key ZDOS 64180 to be exact.

I have just recently read an article (Computer Design/Feb. 15, 1987) covering all single board computers, and most are not offered as complete systems. There are two reasons for this, lazyness (more accurately; hard work with little return) and users' needs. While working at a S-100 company, I tried to get complete systems offered through us. The owner decided the extra work would not match possible outlays. They also said "we manufacture BOARDS for people who sell systems."

In the case of the STD Bus, these products reflect the industrial users' needs and not general consumers interests. I feel one of the strong selling points of the STD Bus is the manufacturer's feeling that you should integrate your own system. This may mean more work for you, but the result is a product that meets your own special uses. Should you want a system that can do regular computing, buy a regular computer. If you want a fast machine for a single job, buy SDT Bus cards and make your own system.

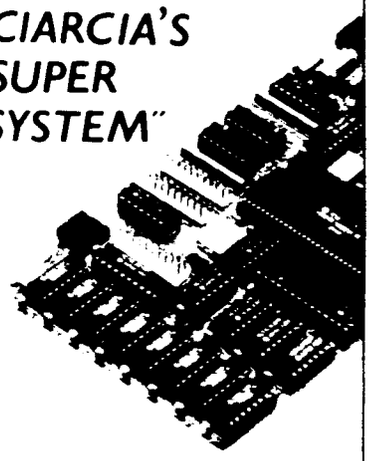
What you are really saying, and this is very true, is that more complex CPUs can make too complex a system for most people to put together. This is the reasoning that got me into Forth and Hawthorne Technology's 68K operating system. A good way to check this is to see what is being manufactured by the 40 STD Bus makers. The article in Computer Design has a table showing eight speciality cards (8032, 8052, TMS9995, TMS32020), 11 others using 68Ks, 14 for the 6809 (1 65F11, Forth on a chip). The PC class cards were 32 types of 8088, 80188/6/7, V20, V50. The big winner however was 65 cards of the Z80 and 8085 type. Your interest in the 64180 has only 4 cards being made, 2 from one company. That is 136 cards, almost half of which are the easy to program and fast Z80. I called all three 64180 people, one is not in production yet (very iffy), one was a special project, and one company has it for sale and thinks you could use their CP/M (no extra memory access). Which, if all you are interested in is the speed, pull your Z80 and watch the program scream at 9.2 MHz on the 64180 (using old CP/M). That is what I like about the STD (or any good bus system), the ability to keep your I/O and just upgrade the CPU or memory.

There are two questions that these facts don't cover, how much memory does the average STD system use, and how many are running programs written 4 or 5 years ago. I feel one of the good points which ZDOS and CP/M are good examples of, is just how much you can get done in 64K of memory. I doubt that many industrial applications need the larger memory of the PC cards. Those PC cards are used, I guess by the newer programmers who have only learned on PC monsters and could care less about fast assembly language (this was validated by one of the tech people I talked to and some other articles listing the new "C" compliers for speciality CPU chips). I will admit that there are some new applications that could use the bigger more complex multiuser options now available on the STD Bus, but as a whole the U.S. is behind in automation. What I mean by this, is that I have not seen enough new applications that would

(Continued on page 40)

Byte Magazine called it.

"CIARCIA'S SUPER SYSTEM"



The SB180 Single Board Computer

Featured on the cover of Byte Sept 1985, the SB180 lets CP/M users upgrade to a fast, 4" x 7½" single board system

- **6MHz 64180 CPU**
  (Z80 instruction superset), 256K RAM, 8K Monitor ROM with device test, disk format, read/write
- **Mini/Micro Floppy Controller**
  (1-4 drives, Single/Double Density, 1-2 sided, 40/77/80 track 3½" 5¼" and 8" drives)
- **Measures 4" x 7½" with mounting holes**
- **One Centronics Printer Port**
- **Two RS232C Serial Ports**
  (75-19,200 baud with console port auto-baud rate select)
- **ZCPR3 (CP/M 2.2/3 compatible)**
- **Multiple disk formats supported**
- **Menu-based system customization**

New Low Prices

SB180-1
SB180 computer board w/256K bytes RAM and ROM monitor $299.00

SB180-1-20
same as above w/ZCPR3, ZRDOS and BIOS source $399.00

COMM180-S
SCSI interface $150.00

Now Available

TURBO MODULA-2.....$69.00
TURBO MODULA-2 with Graphix Toolbox.....$89.00

TO ORDER CALL TOLL FREE 1-800-635-3355

TELEX 643331

For Technical Information or in CT, call **1-203-871-6170**

Micromint, Inc.
4 Park Street
Vernon, CT 06066

Starting Your Own BBS

What it Takes To Run a Bulletin Board

by R.E. McCain

So you want to run an Electronic Bulletin Board System? Once you commit yourself to having an operational system "on-line" you find that you are on a homemade raft floating down a river. When the stream is wide and sluggish it seems that you are not going anywhere. When you reach the rapids things go by so fast that you loose track of who you are, where you are going, and run the risk of drowning!

I know that it may sound like the same old saw everyone keeps pounding in your ears, but go slow. Keep your eyes and ears open to your surroundings. A small slip up at the beginning can cause endless hours of frustration later.

So lets start with the assumption that you, your spouse, your company or club wants an operating BBS.

- What are your present resources?
- How skilled are you at programming
- How much can you afford to spend?
- How much time will you have to set up the system?
- How much time will you have to maintain the system?

Let's address these issues before we even look at the mechanics of a working BBS, since everything hinges on the answers.

WHAT ARE YOUR PRESENT RESOURCES? Do you have a spare computer hanging around? How standard is it? (I.E. CP/M, MS-DOS, APPLE, ATARI, COMMODORE) If it is a prototype of a Monster 9001 that Uncle Louie left you in his will, you have NO DOCUMENTATION and not the slightest idea how it works—FORGET IT! How much software do you have for it? NOT how many games, but really useful stuff like BASIC, PASCAL, C, etc. and their respective compilers, and while we are at it, utilities, assemblers, editors, and so on.

HOW SKILLED ARE YOU AT PROGRAMING? I am not asking whether you are a software guru, reknown'd for your comprehension of the inner mysteries of UNIX, or some similar stuff... just how comfortable are you with simple things like the basic command syntax of your "target" computer. Do you recognize the A> in CP/M? Does the word PIP mean anything more than a Charles Dickens character? If you want to look at the contents of a disk do you type DIR, CATALOG, or "ls -l"? If little of this makes sense, head off in one of two directions: go find a GOOD software supplier who will provide you with a turnkey system, or prepare yourself to learn a LOT about what makes your machine tick!!

HOW MUCH CAN YOU AFFORD TO SPEND? This seems to be a good time to look at the two strategies of setting up a BBS. If you have a deep pocket (or friends in high places) you can get a system that will do just about everything but put out a cigarette! (With a little robotics coupled with an infrared camera you too could have a more positive way to remind others not to smoke.) Beware the Jabberwocky salesman who wants to sell you a VAX driven by a little old UNIX clone! Shop around, become an informed consumer before you pull out your plastic. Look at the software now available, do NOT trust VAPORWARE PROMISES!

The other (and possibly more practical for those who have no large bank accounts) method is the component assembly technique. You purchase a CPU (this is not 1960, an AMPRO Little board or equivalent is just fine, or you could find one of the many IBM-PC clones), a floppy drive or two (the more storage space the better), and a display device. Get the thing running and learn its quirks and capabilities, then add on a modem. Get the fastest auto answer you can find that has Bell 103A and 212A compatibility, smartmodems are now so cheap it makes sense to get one, rather than an old dumb clunker. As time goes by, you can add a hard disk or two, a printer, etc. When you are ready to test the BBS software, use your home phone line to verify operation, but have a separate line put in for the BBS before you start advertising!!! The $10 a month for the second line is well worth it... in ulcer reductions alone!

HOW MUCH TIME WILL YOU HAVE TO SET UP THE SYSTEM? This one is the old time / money / involvement (to the exclusion of all else) equation. Never expect everything to work as planned! A simple BBS with a purchased turnkey software package on the exact machine specified may work within the first hour.... or it may take months (if some manufacturer changed something and didn't tell anyone!). If you enjoy puttering, you can have "something up" fairly quickly and spend the rest of your life improving it.

HOW MUCH TIME WILL YOU HAVE TO MAINTAIN THE SYSTEM? Some systems require almost daily nursing and feeding, while others can take care of themselves. On a small system (under a couple of megs of disk space) you have to do a lot of housekeeping or else you run out of disk space quickly. The type of BBS you choose will also determine how closely you must monitor things. A closed (no unverified callers allowed) system is less subject to gatecrashers. An open forum where no one can enter the operating system provides "ragchewing" but offers little for software exchange. A software exchange must constantly be on guard for those who would upload copywritten or proprietary software for illegal exchanges, as well as for the pranksters who upload hard-disk killer stuff.

This leads us to a discussion on what it is that you want your BBS to be able to do for you and your users. The primary uses for a BBS are:

1) Networking for those of similar interests.

2) Software exchange of public domain or "shareware"programs.

3) A business or club desiring to connect field personnel to a central location.

4) a shared resource, such as a database.

5) Passing the time (ragchewing—an old Amateur Radio word)

6) Providing a problem solving/informative service for customers or friends.

7) No doubt many other ideas will occur as time marches on..

Notice how many of the above blend into each other... What was once considered a frivolity (Amateur Radio for example) is a national resource in times of trouble.

So you ask, what are the major components of a BBS?

1) Communication with the outside world—a modem.

2) Information storage and retrieval—memory, disk drives.

3) An executive to direct the process—the software.

4) An engine to do the work—the computer.

5) Documentation! Get all the manuals, write everything down, you will need it six months from now when something traumatic happens (Spilled cola on the only boot floppy or installing a hard disk, etc.).

All of these can be in as small a package as a TRS80-100, or as large as a CRAY2 complex. I do not recommend either extreme!

A good size for a BBS is a typical CP/M or MS(PC)-DOS machine. An old IMSAI with hard sectored Micropolis 96 tpi drives might only be servicable for a short while, since replacement parts are getting scarce, but might be enough to get started. Public domain software is usually written for CP/M or MS-DOS. The Apple][series and Macintosh are sort of non-standard in this area, as are Atari and Commodores. However, you can add the CP/M options (which are often kludges I do not recommend for something that must be reliable for months or years).

Lest I neglect to mention it anywhere else, advertising is, and must be, a key element in setting up your system. You must somehow inform the folks that you want to use your BBS of its existence! Anything from word of mouth to billboards is acceptable. For example, my BBS gets a free plug in both "Computer Currents" and "Microtimes" in every issue. Many trade, professional, hobby and specialty publications are willing to give space to "networking activities."

Most of the public domain software available for BBS use is written for Apple, CP/M or MS-DOS compatible, and I recommend the Public domain stuff highly, mainly because it seems to be far more reliable and (surprisingly) better documented and supported than the Commercial packages! The major hangups with most public domain stuff are that many of them assume that you have such exotica as BASIC, C, or Pascal compilers. This means you do have to spend some money after all... but, it also gives you the freedom to CHANGE the things you do not like!

A few of the Public Domain BBS packages have recently simplified things a bit. Programs such as MBBS and RBBS3 now have pre-compiled software (both are written in Microsoft Basic, which has a compiler (BASCOM) which was never designed for long and complex programing efforts). While this does simplify matters somewhat, it removes a little of the flexibility. So RBBS3 also supplies the source code, for those with the inclination to modify beyond the options offered in the install program.

However, few of the BBS programs live alone! Most require BYE to answer the phone. While BYE can be a standalone for a very small scale system, it usually hooks into either a message system (MBBS, METAL, OXGATE, PBBS, RBBS, ZBBS and the like) or a shell of some sort (such as XMENU). Most Message systems provide for user logins, passwords, keep track of mail, provide bulletins, news, help, feedback for the operator, chat and so on.

There is usually a way to get out of the message system into the operating system of the host computer. BYE will typically filter out nasty control characters, hide system files, deny downloads of .COM files (rename the ones you want to be downloadable as .OBJ), deny uploads of .NDR, .ENV, .NDX, and .RCP files (for ZCPR3 systems) and keep track of the users time on the system. The caller may then function in very much the same way as with their home computer. One can up and download files, view text files, look at directories (both of user areas and of .LBR files), and find out what resources the system has to offer (FOR, NEW, SYSMAP etc.). Many of these functions could be combined in a shell with menus to guide the novice through the learning process, with options to enter the real "COMMAND" level at the discretion of the SYSOP (that is the person who runs the board—i.e., you).

I'd like to clear up some possible confusions about the names and functions of various pieces of software I've mentioned. There are at least 3 versions of RBBS, RBBS3 is written in Microsoft Basic, RBBS4 in BDS C, and there is also RBBS-PC for the IBM-PC. KMD is starting to replace (or at least co-exist with) XMODEM as a block transfer program. BYE5 was written to merge the features of MBYE and BYE3. PBBS is written in assembly language, which might be ideal, except that M80 or an equivalent macro assembler is required. ROS (written in Turbo Pascal) and RBBS-PC are both stand alones that include the BYE and XMODEM functions internally.

Our BBS system is typical of a "Homebrew BBS." It has an AMPRO 1B Main board with SCSI bus, a 6 meg Shugart hard disk with a Xebec 1410 SCSI controller, a Shugart 455 48tpi and a Shugart 465 96tpi. The 48tpi drive enables us to read/write most other formats and the 96tpi is for backup. It uses an auto answer 300/1200 baud VenTel modem (a dumb modem, although funds are being raised for a smart 3/12/2400 modem) and is housed in a modified surplus TeleVideo case.

We started out with a homebuilt "tower" sitting on the floor under a desk. For the first few months we used 3 surplus Shugart 410 96tpi single sided floppies (the worst model that they ever built—notoriously unreliable). There were discrete linear power supplies in the bottom which made it stable (but warm), and a cooling fan (with a filter that had to be washed at least once a month). The linear supplies were cheap, the fan is needed for the reliability one expects from a BBS. We use a second hand ADDS Viewpoint terminal with the bell muted and the intensity turned down. Since the terminal is on 12 to 24 hours a day, it is better not to burn the phosphor prematurely. We use BYE5, RBBS3, KMD, and XMENU, but could just as easily use BYE3, PBBS, XMODEM and allow access directly to C/PM.

The one area that the SYSOP cannot skimp on is MAKING REGULAR BACKUPS OF THE ENTIRE SYSTEM. There will inevitably come a time when the disks crash, you are attempting to do housekeeping and run out of disk space, or some other disaster strikes! It is much easier to enter a general bulletin to the effect that you had to restore last weeks backup, will everyone please check and see if they need to re-enter a message, etc. than to begin the entire process of rebuilding from scratch.

Since we ran our BBS for 9 months before the hard disk went

on line, we can survive a hard disk crash by putting the floppy backup set in, and re-booting. We may not have all the resources because our disk space just dropped from 7 meg to 1.2 meg, but we are still online. Plan for such contingencies! Nothing is so demoralizing to your users than a BBS that is always down! (And to you too, for that matter!)

There has been a lot of concern about "gate-crashers," i.e. hackers whose greatest joy in life is crashing things like BBS systems. I have found that many of the problems encountered in the process of setting up and running a BBS are blamed on such folks, but closer examination often reveals "user error," "SYSOP error," a floppy at the end of its life, or noisy phone lines. Use your troubleshooting skills. If you really suspect a "hacker" at work, set the BYE option that saves every character entered by the remote user in a log. That will give you a clear audit trail of the circumstances of the failure. You most likely will find that any time you have "improved" the system in some way, you are the guilty party! Never make a major system change without checking things thoroughly!

Troubleshooting is most easily accomplished by setting up another computer or a terminal and modem up on your voice phone line to call the BBS phone line. (You did install a second line?) This enables you to easily emulate the average user and see what happens from both ends simultaneously. If you cannot do this because you don't have the resources, the second best thing is to arrange with a friend who has a modem and computer to call up the board and capture everything so you can review it later. Try using machines wih different screen sizes. It is amazing how different your menus written on an 80 column display look on a 40 TV screen!

As time goes on, you will find that the variations in keyboards, displays, modems, and software used by your callers will create little annoyances that you need to resolve (usually to the lowest common denominator). Should you be blessed with a situation where all your users have identical stuff, you may ignore this paragraph, otherwise be forewarned! Text files written on a Macintosh word processor usually come out as garbage on a machine that expects an ASCII text file. Other text files (such as those from Wordstar) may need to have the 8th bit stripped off. On the other hand, some Commodores go into graphics mode with the 8th bit off, while PCs go into graphics mode with the 8th bit set. These problems most often come up during XMODEM transfers, where an exact image is transmitted. ASCII dumps and saves may be better for text files. You and your users will have to work out solutions to these problems.

Some machines (the old Apple II for example) do not produce certain ASCII characters without some sort of kludge. Stay away from those characters when you generate one key commands! Some communications software, such as CrossTalk, uses the ESC key for local command initiation. Is it fair to the caller to have to change all their default parameters so they can use your BBS?

If the above has given rise to any thoughts about forgetting any plans you had for implementing a BBS, I apologize. I've had a lot of fun getting mine to work (though I've had many frustrations—mostly because I tried to rush things) and I enjoy the daily challenges of keeping it online, interacting with the callers, and watching the screen as some new user discovers the nooks and crannys built into the system. ■

EVERYTHING YOU NEED . . . $279.00
Now it's easy to program the Heath-Zenith HERO-1® Robot with an Apple® II. HERO® Macros for the S-C Software 6800 Cross Assembler program in Heath's Robot Interpreter Language with easily remembered mnemonics. The HERO® Macros come with 30 pages of documentation.
[For example, the line 1130 > MVWRIM GRIP, OPEN, 60, FAST instructs the HERO® to open his gripper 60 units at fast speed. Motor position is expressed in base-10.]
Transfer to HERO® with ROBI . . . an affordable interface for the robotics experimenter . . . is simple.
• ROBI is a complete package. No additional hardware required for Apple® or HERO®.
• ROBI installs quickly in an Apple® II, II+, or IIe. Once installed, no hardware changes are needed. Within minutes, you will be programming HERO®.
• With ROBI and the Cross Assembler, the programmer uses Apple®'s memory to write the program, and HERO®'s memory to run the program.
• Not "copy protected," archival copies may be made as needed.
• ROBI offers expansion potential.
VISA and MasterCard accepted.
BERSEARCH
Information Services
26160 Edelweiss Circle
Evergreen, Colorado 80439
The Cross Assembler with HERO® Macros sells for $100.00; the ROBI Interface sells for $199.00. Both as a package — $279.00.
To order, or for more information, call (303) 670-6137.

Build an A/D Converter for the Ampro L.B.

A Simple Low-Cost A/D Interface

by John B. Rasor, D.O.,

This article describes a simple, easy to install analog to digital converter for the Ampro Little Board. I developed it for acquiring biomedical data, so it is well suited for digitizing and isolating noisy, low-frequency signals. The interface requires only a single clip-on connection, so it should be useful for those limited by the single input port available on an unexpanded Little Board. Only a single 14 pin chip is necessary.

The actual analog to digital conversion is performed by a TSC9400CJ Voltage to Frequency convertor. The pulse stream is connected to one or two of the spare Z80 Counter/Timer (CTC) ports on the Little Board. The "B" serial port is used by the software for timing sampling intervals. At the end of one of these intervals, the corrected value in the Counter/Timer register(s) is directly proportional to the average input voltage during the interval. This is an integrating analog to digital technique, which is useful for reducing noise from the input signal. It also allows very high resolution if long enough sample times are used. Signal isolation or long-distance transmission are easily done on the pulse stream between the interface and the Little Board.

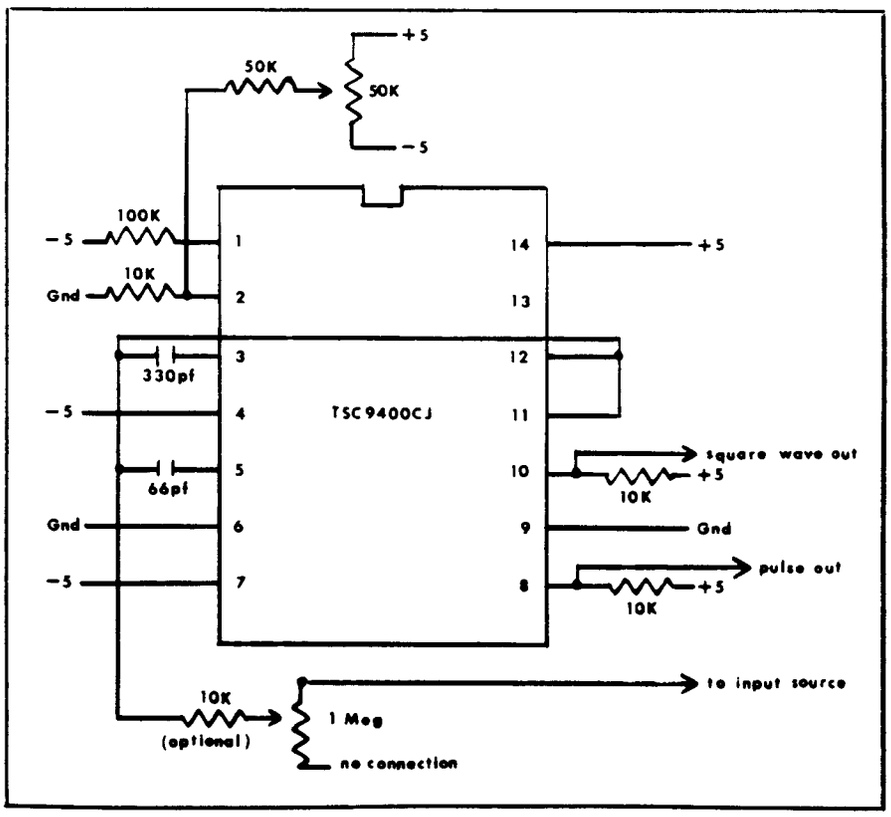
The TSC9400CJ is a useful chip. Unfortunately, Radio Shack no longer carries it. It is made by Teledyne, and is available from the address given at the end of this article. A full range input requires only 10uA of signal. This can save you a stage of gain or two. For full 0-5 volt range signals, the optional 100K input resistor limits the input current to the specified maximum of 50uA. The capacitor values are not at all critical, but the larger should be between three and ten times the value of the smaller. Pin 8 is a pulse output, while pin 10 is a square-wave at one half the same frequency. Either is suitable for driving the Little Board's CTC port.

The output of the TSC9400CJ chip is a series of pulses or square waves at varying frequencies. It can be isolated or buffered as needed before connecting it to pin 21 of the Little Board CTC chip. This is the counter trigger for the unused CTC2 channel (CLK/TRG2). CTC3 is used only for allowing interrupts from the floppy disk controller, a feature which is not supported by any software I am familiar with. Therefore, I cut the trace from CTC pin 20 and FDC pin 28 between two plated-through holes on the top of the board. This allows 16 bit counts to accumulate, which makes the software simpler and better performing. The same software can be used without cutting the trace, but the user must insure that the timing interval is short enough to prevent overflowing the 8 bit counter.

The output frequency is directly proportional to the input current, and inversely proportional to the sum of the smaller capacitor plus an internal 12pf capacitor. This design gives a theoretical maximum frequency of 128 khz. In practice, I get about 98.5 khz for the full 50ua input.

Since only a single external connection to the board is required, I used a removable IC-type micro-clip. Otherwise it would be necessary to solder to the chip lead. If you use the 16 bit counter scheme, another jumper needs to either be soldered or clipped from CTC pin 9 (ZC/TO2) to CTC pin 20 (CLK/TRG3) to cascade the two counters. Ampro's Time and Date commands will work with these jumpers in place (but not while they are in use!) A connection to the system Ground also needs to be made somewhere. Usually the power supply or case is more convenient than the CTC or other chip.

The Z80 CTC chip contains four individual channels that may be used either as counters or timers. Except for the lack of an external output signal from channel 3, they are identical. Each channel must be initiallized before using it. This is done by issuing a combination reset and chan-



nel control command, and then a time constant for each channel. This application uses the counter mode (without interrupts) triggered by the rising edge of the input signal. A timing constant word of zero is specified. The two byte sequence that needs to be sent to the channels used is thus: $57 (the reset command and channel control word), $00 (the timing constant). Once the timing constant is written, the channel will respond to each low to high transition of its trigger line (CLK/TRG pin) by decrementing the count. The reset command stops the count in progress for the channel it is issued to. A read of the channel at any time returns the present value of the count.

The software must compensate for some of the limitations of the Z80 CTC chip. For one thing, there is no explicit means of synchronizing the startup or reading of two channels. For a cascaded system, the software should reset both channels first, then write the more significant channel's time constant. It can then start the less significant channel. This technique prevents the less significant counter from counting down to zero and triggering the more significant counter during the initiallization sequence. Both counters are thus effectively started off from zero. Before reading the counters it is necessary to stop the less significant one from counting by issuing a reset command. Since the counters count downward, the actual count is the two's complement of the 8 or 16 bit value read from the counter's registers.

The Little Board serial port "B" is used to time the sampling intervals. It receives its timing signals from the CTC channel 1 in timer mode. This method sends a steady stream of nulls out of the serial port. Fortunately, these do not upset either my modem or floor-crawling robot, the two peripherals that fight for the use of the overworked "B" port. Perhaps some day I will add several DART's to one of the new expansion boards and have enough ports for all.

The serial port could be left alone by just using the CTC channel 1 for timing and leaving the DART out of it. The only problem is that a single CTC channel connected to a 2 megahertz signal (like CTC1 is) can only generate an interval of 32.768 ms or shorter. This is not long enough for my application, although it may be for yours. This also requires calculating the proper parameters and programming the CTC channel. Using the DART set up for 8 bits, 1 stop bit, no parity, and no handshaking generates samples that are exactly one tenth the baud rate (30 samples per second for 300 baud, etc.) The sampling rate can then be easily set from the DOS by using the "SET" command.

Using the DART this way is reasonably straightforward. A character is sent to the DART for transmission and the counters started. The DART is polled until it indicates it is again ready to send. The data is then read from the CTC channels and the cycle is repeated. The only trick here is that the DART doubly buffers data to be transmitted. This means that the first two intervals are not accurately timed because the DART is not yet "saturated" with data. I obtain two extra data samples and then discard the first two to avoid this problem.

The driving software can be written in any reasonable programming language. Interpreted BASIC results in lesser performance, but will run up to about 30 samples per second. I use the BASIC compiler, which is quite adequate for 960 samples per second and probably would go quite a bit faster. The software includes a timing test to see how well the system can time intervals for a given rate. This software is a subset of a larger package that also provides signal processing, Fourier analysis, signal storage and retrival, and biofeedback information on electrocardiograph (heart) or electroencephalograph (brain) signals.

Two examples show both the usefulness and problems with this sort of integrating A->D convertor. Figure 1 is a

AMPRO A/D LISTING

```
1 ID$="ARTICLE.BAS V04 19 February 87"
2 DEFINT A-Z
5 N=32:DIM X1(1024),X2(1024):WW=60:PRE=2
20 DEF FNB(X)=255 AND 256-X
30 GOSUB 10000
35 INPUT "Would you like a timing test";A$:IF LEFT$(A$,1)="Y" THEN GOSUB 5000
40 GOSUB 12000:GOTO 40
540 REM Display data in X1 array.
545 REM Removes DC offset and scales data to the display size.
550 FOR Z=0 TO N-1
560 IF ABS(X1(Z))>B THEN B=ABS(X1(Z))
570 IF X1(Z)<C THEN C=X1(Z)
580 NEXT Z
590 IF B=C THEN PRINT"didn't receive any data, will try again":RETURN
600 FOR Z=0 TO N-1
610 PRINT X1(Z);TAB(8+WW*(X1(Z)-C)/(B-C));"O"
620 NEXT Z
623 PRINT"Max value=";B;", min value=";C
625 RETURN
5000 REM Timing test
5010 INPUT "What is the ideal sampling rate";RATE
5020 PRINT"Press any key and begin your timing - should finish in 1 minute."
5030 A$=INPUT$(1)
5040 Z=RATE*60
5050 FOR I=1 TO Z:
        OUT DARTBDAT,0:
        WAIT DARTBCON,DART.TXEMPTY:
        NEXT I
5060 PRINT "All done.":INPUT "How many seconds was that";I
5070 PRINT "Actual rate=";Z/I,"Difference=";(ABS(RATE-(Z/I))/RATE)*100;"%"
5080 RETURN
10000 REM Special initialization for using a->d
10020 CTC2=&H60:CTC3=&H70:CTC.CCW=&H57
10030 DARTBCON=&H8C:DARTBDAT=&H88:DART.TXEMPTY=4
10040 RETURN
12000 REM Load real data from a->d and process it
12010 GOSUB 13000
12030 GOSUB 540:GOTO 12000
13000 REM Read N samples from a->d
13005 PRINT"Sampling..."
13010 M=N-1:L=M+PRE
13015 OUT CTC2,CTC.CCW
13020 FOR A=0 TO L:
        OUT CTC3,CTC.CCW:OUT CTC3,0:OUT CTC2,0:
        OUT DARTBDAT,0:WAIT DARTBCON,DART.TXEMPTY:
        OUT CTC2,CTC.CCW:OUT CTC2,0:
        X1(A)=INP(CTC3):X2(A)=INP(CTC2):
        NEXT
13025 PRINT"Computing..."
13030 FOR A=0 TO M:
        X1(A)=256*FNB(X1(A+PRE))+FNB(X2(A+PRE)):X2(A)=0:
      NEXT A
13040 RETURN
15000 REM Goto this line for a continuous display of the CTC inputs.
15010   OUT CTC2,CTC.CCW:OUT CTC3,CTC.CCW:OUT CTC3,0:OUT CTC2,0:
        OUT DARTBDAT,0:WAIT DARTBCON,DART.TXEMPTY:
        OUT CTC2,CTC.CCW:OUT CTC2,0:
        X1(A)=INP(CTC3):X2(A)=INP(CTC2):
        PRINT "CTC2=";INP(CTC2);"CTC3=";INP(CTC3);CHR$(13);:
        GOTO 15000
```

Figure 1

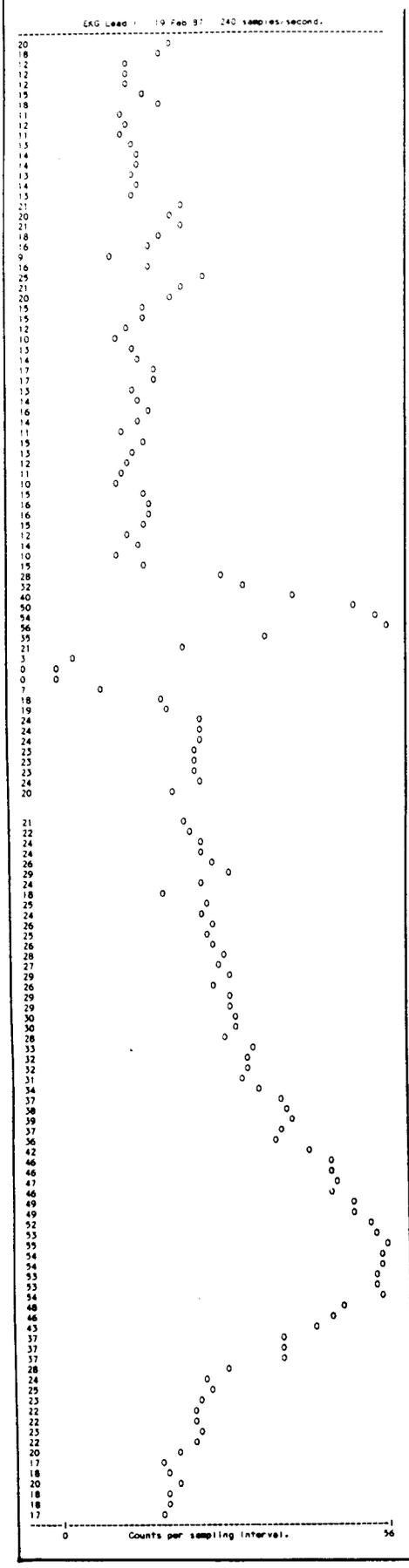
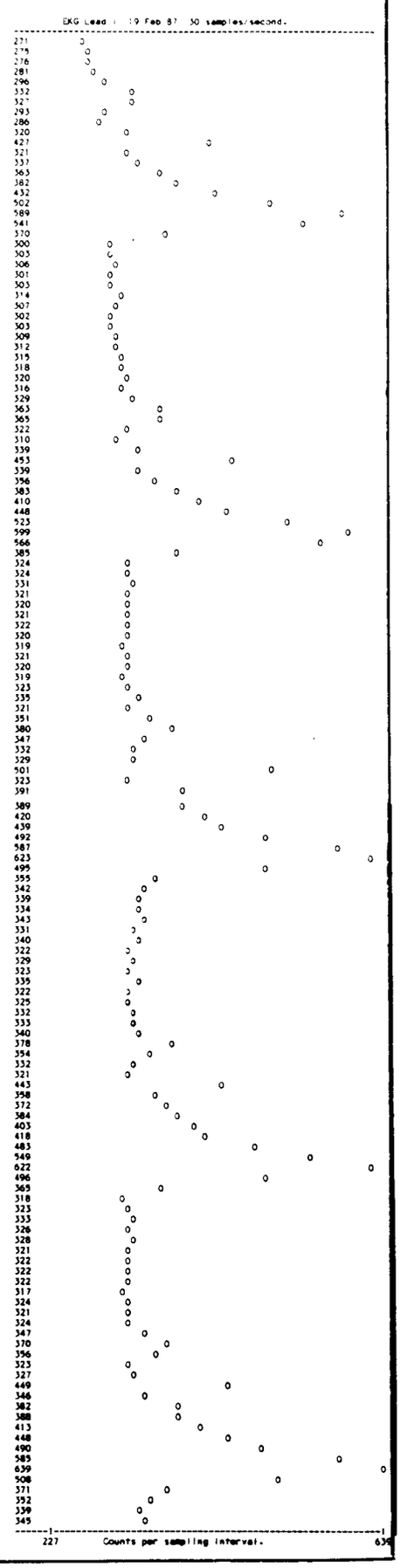


Figure 2



fragment of an EKG obtained at 240 samples per second. High frequency noise is present. Figure 2 is part of an EKG obtained at 30 samples per second. The baseline noise has been averaged out and the EKG signal is more clearly seen. The signal is, however, slightly distorted by the averaging process and unsuitable for detailed medical interpretation.

The chief drawback to this method of analog to digital conversion is its poor performance at increasing frequencies. At a 20 kilohertz sampling rate, say for speech processing, you would barely get 2 bits of resolution. Of course, the converse is also true, and resolution at low sampling rates can be as good as the 13 bits the TSC9400CJ chip is capable of delivering. Let me know of any interesting applications you come up with!

The TSC9400CJ chip can be obtained from:

Marshall Industries
9674 Telstar Avenue
El Monte, California 91731

The cost was $4-$5 as of February '87. Their toll-free order line is 1-800-522-0084 (outside of CA.). ■

$45 MORROW CP/M ENGINE

- Over 15,000 of these single board computers installed.
- Free copy of the "MORROW OWNERS REVIEW" The national magazine devoted just to the MORROW computer. while supplies last
- We have over 2000 in stock
- 4 mhz Z80A CPU, 64K RAM 16/32K ROM
- 2 RS232 serial ports (300-19.2k baud with db25 connectors installed)
- Centronics printer port cable extra $8
- Power requirements- +12vdc -12vdc +5vdc
- Floppy disk controller up to 4 drives. SSDD standard. Rom for DSDD 48tpi or 96tpi $12 extra including assembled bios
- CP/M bios, Wordstar, and Bazic included
- Schematic, bios.asm, maint manual and users guide included
- Copy program to read/write non morrow formats such as Kaypro, Osborne, Xerox, & etc.
- Optional rs232 terminal pcb, TTL monitor pcb, and display tube $59 (with purchase of cpu, a 30% savings from seperate price)

Call fo a copy of 15 day trial agreement. Tax & freight extra. Send check or add 1.90 for COD. Price may change. Store price may differ. While supplies last. No POs, terms, or credit cards

Silicon Valley Surplus
415-261-4506
4401 OAKPORT OAKLAND CA, 94601

OPEN 10am-6pm
CLOSED SUN & MON

The Hitachi HD64180

New Life for 8-bit Systems — Part 2

by Jon Schneider

In the first article of this two part series, the only advanced feature of the Hitachi 64180 microprocessor to be addressed in detail was memory management. This installment will cover setting of wait states and RAM refresh rates, the use of the PRTs (Programmable Reload Timers), and will touch briefly on the chips DMA (Direct Memory Access) capabilities.

Wait States and RAM Refresh

Since wait states and RAM refresh rates are usually the province of hardware designers, and not a function that programmers are accustomed to having control over, a brief discussion of their purpose is in order.

Most of the RAM used in personal computers today is dynamic RAM, and due to its design, it must be periodically accessed or its contents are lost. In a Z-80 based system, the RAM refresh is performed after every op-code fetch, and is solely dependent on the system's clock speed. Since refreshing of memory is handled by the microprocessor, as the refresh rate is increased, the effective throughput of the chip decreases.

Wait states are merely the insertion of do-nothing T-states between the placement of an address on the address bus, and the reading or writing of data on the bus. These are added to insure that slower devices (either I/O or slow memory) are not out-run by a fast micro-processor.

Since the Hitachi processor was designed with the idea of being able to add it to existing hardware, the capability to have software control over RAM refresh rates and wait state generation was included as an on-chip function. This allowed the chip to run with just about any pre-existing hardware, and still have the ability to run with newer and faster hardware without having to spend too much time refreshing memory or adding to many wait states.

Wait states for both I/O and memory are controlled by the same register. The four MSBs of the register are used for wait states, and the 4 LSBs are used for setting up DMA transfers, as shown in Figure 1.

Regardless of the I/O wait state setting, all on chip ports are accessed with 0 wait states. For on chip serial port data register accesses and PRT access, 1 to 4 wait states are added.

If you are writing a BIOS, or the firmware for a 64180 based controller, you will have to determine the optimum values for wait state insertion, and initialize the register after the system is powered on or reset. This is necessary, as when the chip is reset, the values are set to the maximum number of wait states possible. More than likely, at 6 mhz and with 150 ns memory, you will be able to run with 0 memory wait states. The number of I/O wait states is dependent on what type of I/O devices you will be running.

If you are writing applications programs, then you will have no need to alter the DCNTL register, unless you are utilizing DMA, in which case you will need to make sure that you do not alter its 4 MSBs when you change its 4 LSBs for setting up the DMA transfer. This will be covered later.

Figure 1

DMA/WAIT Control Register (DCNTL : I/O Address = 32H)

```
bit  7    6    5    4    3    2    1    0
     MWI1 MWI0 IWI1 IWI0 DMS1 DMS0 DIM1 DIM0
```

MWI1 - MWI0 (bits 7-6)
MWI defines the wait states that will be inserted for CPU and DMAC cycles that access memory (including memory mapped I/O). 0 to 3 wait states are automatically inserted depending on the value programmed. After a hardware reset all of the MW bits are set to 1.

MWI1	MWI0	Number of wait states
0	0	0
0	1	1
1	0	2
1	1	3

IWI1 - IWI0 (bits 5-4)
IWI defines the wait states that will be inserted for CPU and DMA cycles that access external I/O (and interrupt acknowledge cycles). 1 to 6 wait states are automatically inserted depending on the value programmed. After a hardware reset all of the MW bits are set to 1.

IWI1	IWI2	Number of wait states: For external I/O	Number of wait states: For INT0 acknowledge cycles when LIR is low
0	0	1	2
0	1	2	4
1	0	3	5
1	1	4	6

DMS1 - DIM0 (bits 3-0)
Utilized for setting up DMA transfers. Will be covered later.

DMA Controller

The Hitachi 64180 includes a very versatile two channel DMA controller on the chip, and allows the following mode transfers;

```
Memory <------> Memory
Memory <------> I/O
Memory <------> Memory mapped I/O
Memory address increment, decrement, no-change
Burst or Cycle steal Memory <---> Memory transfers
DMA to or from both ASCI channels
```

DMA channel 0 allows all of the above modes, but DMA channel 1 only allows Memory <----> I/O transfers with address increment or decrement. Channel 0 also has a higher priority than channel 1. Both channels signal the end of a transfer by by pulling the TEND (Transfer END) line low, setting a bit in a status register, and optionally, generating an interrupt.

Since the DMA circuitry has access to the Hitachi's full address bus, the DMA transfers can be made to any address within the 512kbyte address space without the necessity of altering any of the bank selection registers. The only way that the 64k limit applies is that the transfer length is limited to that size. That is due to the fact that the byte count register is 16 bits wide.

A DMA byte transfer can occur every 6 clock cycles, and wait states can be added to compensate for slow memory or I/O devices. At a clock rate of 6 mhz, with 0 wait states, transfers can occur at the rate of 1 megabyte per second.

There are 19 registers that pertain to DMA transfers, and there are many rules to follow when setting them. It is beyond the scope of this article to cover them all. It is an absolute necessity to have the Hitachi manual as a reference when programming for DMA transfers, and even then it is confusing. For that reason, I have included two sample sections of code in Listings 1 and 2 that use DMA. You will need to have the Hitachi manual in order to fully understand the code.

One instruction you may notice in the DMA code that has not been covered previously is OTIMR (Out with Increment, Repeat). It is essentially the same as the Z80 OTIR instruction, except it assures that all high order address lines are zero, guaranteeing access to the Hitachi's internal ports. The same similarity exists between OTDMR and OTDR, OTIM and OUTI, and OTDM and OUTD.

Programmable Reload Timers (PRTs)

The Hitachi 64180 has a two channel programmable reload timer, and each channel has its own 16 bit reload register and 16 bit timer data register. Both channels can be programmed to generate an interrupt request.

The incorporation of on-chip timers facilitates the use of a BIOS real time clock, device time-out routines, a flashing cursor, and various other real time routines. The only variable that has to be taken into account is the speed of the processor, as the timer data register is decremented once every 20 clock cycles.

In addition to the obvious uses, PRT channel 1 has an output pin dedicated to it (TOUT) that allows the Hitachi to perform programmable output waveform generation. I will not cover its use, as I have never found a need to use it, and doubt if many other programmers would either.

To utilize the PRT, it is only necessary to set up two registers, the Timer Control Register (TCR), and the Timer Reload Register for the desired channel (RLDR).

```
Timer Control Register (TCR : I/O Address = 10H)

     bit 7    6    5    4    3    2    1    0
         TIF1 TIF0 TIE1 TIE0 TOC1 TOC0 TDE1 TDE0

TIF1 - Timer Interrupt Flag channel 1
     When the timer data register for the channel 1 (TDR1)
     decrements to 0, TIF1 is set to 1. If the timer
     interrupt enable is set to 1 for that channel, an interrupt
     is generated. TIF1 is reset to 0 when both TCR and
     either byte of TMDR1 is read, or on reset.

TIF0 - Timer Interrupt Flag channel 0
     When the timer data register for the channel 0 (TDR0)
     decrements to 0, TIF0 is set to 1. If the timer
     interrupt enable is set to 1 for that channel, an interrupt
     is generated. TIF0 is reset to 0 when both TCR and
     either byte of TMDR0 is read, or on reset.
```

Both of the above bits are read-only. It may seem that they wouldn't be of much use to the programmer, but it is VERY important to understand their use. I didn't at first, and spent the better part of a day trying to find out why my system died after the PRT generated an interrupt.

After a PRT interrupt is generated, the interrupt must be cleared by reading BOTH the Timer Control Register and either byte of the appropriate Timer Data Register. You will see an example of this in listing 3.

```
TIE1 and TIE0 - Timer Interrupt Enable channel 1 and channel 0
     When TIE1 (TIE0) is set to 1, TIF1 (TIF0) will generate an
     interrupt request. If set to 0, the interrupt request is
     disabled. During reset, TIE1 and TIE0 are reset to 0.

TOC0 and TOC1 - Timer Output Control
     These two bits control the use of the TOUT mentioned earlier
     and will not be covered. Just leave them at 0, which is how
     they are after a reset.

TDE1 and TDE0 - Timer Down Count Enable
     When TDE1 (TDE0) is set to 1, downcounting for TMDR1 (TMDR0)
     is enabled for channel 1 (channel 0). When set to 0, it is
     disabled, and when disabled, the respective TMDR can be
     written to. TDE1 and TDE0 are set to 0 on reset.
```

Timer Reload Registers

Both reload timers have 16 bit timer reload registers, accessed as low-byte high-byte. When the Timer Data register counts down to 0, it is automatically reloaded with the contents of RLDRx. It is set to 0 on reset.

The value placed in this register is what determines the rate at which interrupts will occur (if enabled). The programmer will have to determine the proper value to place in this register for the clock speed at which the system is running.

```
RLDROL (Reload Register 0 Low)  : I/O Address = 0EH
RLDROH (Reload Register 0 High) : I/O Address = 0FH
RLDROL (Reload Register 1 Low)  : I/O Address = 16H
RLDROH (Reload Register 1 High) : I/O Address = 17H
```

Timer Data Registers

Both reload timers also have 16 bit data registers. The value in these registers is decremented every 20 clock cycles, and upon reaching 0, is reloaded with the 16 bit value contained in the reload register for that channel. The registers are set to 0 on reset.

The registers are read/write, and can be read without stopping the timer. The proper method to read the data registers is to read the lower byte first, then the higher byte. Anytime that you read the lower byte, the higher byte value is stored in an internal register, and a following higher byte read will read the value from this internal register, not the actual high byte register. This assures the data validity, as even if the register rolls over before the high byte read, the value in the internal register is correct.

To write to the registers, the timer must be stopped by using the TDE bits of the Timer Control Register (covered earlier). Once the timer is stopped, the data can be read or written without any concern as to the order accessed.

```
TMDROL (Timer Data Register 0 Low)  : I/O Address = 0CH
TMDROH (Timer Data Register 0 High) : I/O Address = 0DH
TMDROL (Timer Data Register 1 Low)  : I/O Address = 14H
TMDROH (Timer Data Register 1 High) : I/O Address = 15H
```

Now that you understand the use of the three registers, it is a fairly easy step to see how to utilize them in a BIOS. Let us assume that you want to set up a routine in the BIOS that will allow the use of a flashing cursor, as shown in listing 3.

Listing 1

```
; This code serves as an example of DMA transfer with the 64180
; It is a small section of the low level disk drivers from a BIOS
; modified for a 64180 add-on card.
; It uses the Memory ---> I/O mode on channel 1
; Note that the DCNTL register is modified without altering the
; 4 most significant bits, as mentioned in the section on setting
; up wait states.


; Write a sector
; --------------
fdwtsc: push    hl              ; Save buffer address
        in0     a,(dcntl)       ; Get DCNTL value
        res     1,a
        out0    (dcntl),a       ; Set for memory --> I/O
        call    dmaset          ; Set up DMA
        ld      b,0a0h          ; Set up the write command
        call    fdset           ; Start the command
        dw      fdwts2          ; Termination address for NMI
fdwts1: jr      fdwts1          ; Wait for NMI
fdwts2: pop     hl              ; Restore buffer address
        and     0fch            ; Any errors?

;       continues with error checking


;       called as sub-routine

; Set up for DMA
; --------------
dmaset: ld      a,0c1h          ; Fast refresh
        out0    (rcr),a
        xor     a
        out0    (dstat),a       ; Disable DMA
        ld      hl,dmatbl       ; Beginning of DMA control block
        ld      bc,8*256+marll  ; 8 bytes, channel 1
        otimr                   ; 8 bytes to 8 ports
        ld      a,80h
        out0    (dstat),a       ; Enable DMA
        ret

; DMA control block
; -----------------
dmatbl  dw      hstbuf          ; marll, marlh
        db      0               ; marlb
        dw      fdcdat          ; iarl
        db      0
bytes   dw      512
```

Listing 2

```
; This code serves as an example of DMA transfer with the 64180
; It is a small section of the cold boot routine from a BIOS
; modified for a 64180 add-on card. It simply copies the CCP from
; the system bank to the memory beginning at 256k for use by the
; warm boot routine

; It uses the Memory ---> Memory burst mode on channel 0


        ld      a,80h           ; Set up CBAR
        out0    (cbar),a

        ld      hl,dmatbl       ; Base to HL
        ld      bc,8*256+sar0l  ; 8 bytes starting at SAR0L
        otimr                   ; Out to ports

        ld      a,00000010b     ; mem to mem, increment, burst mode
        out0    (dmode),a

        ld      a,40h           ; enable channel 0 DMA
        out0    (dstat),a

        jp      wboot

dmatbl:
        dw      ccp             ; Source address
        db      0               ; System bank
        dw      0               ; Starting RAM disk address
        db      4               ; RAM disk base bank
        dw      800h            ; 2k
;
```

Listing 3

The following sections of code demonstrate the use of the Hitachi PRT's. The first section of code is from the cold boot routine of a BIOS modified for the 64180, and demonstrates how to initialize the PRT.

```
creset: ld      hl,inttbl       ; Establish interrupt vector
        ld      a,h
        ld      i,a
        out0    (il),l
        ld      hl,(reload0)    ; Initialize timer 0
        out0    (rldr0l),l
        out0    (rldr0h),h
        ld      a,00010001b     ; Enable interrupt and down count
        out0    (tcr),a
        ;continues with cold boot routine
```

```
;the following DW is located in the data area of the BIOS

reload0:dw      30720           ; Timer down count value


    This next section of code demonstrates the handeling of an
interrupt generated by the PRT. The interrupt vector table is
also shown. This routine uses the PRT interrupt to supply a
flashing cursor.


intmsg: db      0dh,0ah,07h
        db      'Interrupt Error',0

; Force a 32 byte boundry for interrupt vector table
; --------------------------------------------------
;       org     $ and 0ffe0h + 20h

inttbl: dw      int1            ; Not supported
        dw      int2            ; Not supported
        dw      inttim0         ; Timer channel 0
        dw      inttim1         ; Timer channel 1
        dw      intdma0         ; DMA channel 0
        dw      intdma1         ; DMA channel 1
        dw      intcsio         ; CSI/O
        dw      intasci0        ; ASCI channel 0
        dw      intasci1        ; ASCI channel 1

; The following interrupts are not supported
; ------------------------------------------
int1:
int2:
inttim1:
intdma0:
intdma1:
intcsio:
intasci0:
intasci1:

interr: ld      hl,intmsg       ; Point to error message
        call    disply          ; Print it
        jp      0000h           ; And warm boot

inttim0:push    hl              ; Save registers
        push    de
        push    bc
        push    af
        in0     a,(tcr)         ; Clear the interrupt
        in0     a,(tmdr0l)
        in0     a,(tmdr0h)
        di                      ; No interrupts
        ld      a,(counter)     ; Get the counter
        dec     a               ; Knock down 1
        ld      (counter),a     ; Store it
        call    z,blink         ; Reverse cursor if at 0
        pop     af              ; Restore registers
        pop     bc
        pop     de
        pop     hl
        ei                      ; Interrupts OK now
        ret
```

You must first determine what channel to utilize. I chose to use channel 0, as that seems to be the one utilized by most 64180 BIOSs, and that leaves channel 1 free for applications programs. Then you determine the frequency at which you want the interrupts to occur, and determine the proper value to load into the reload register. Then load that value into the reload register. You can see how that is done in listing 3.

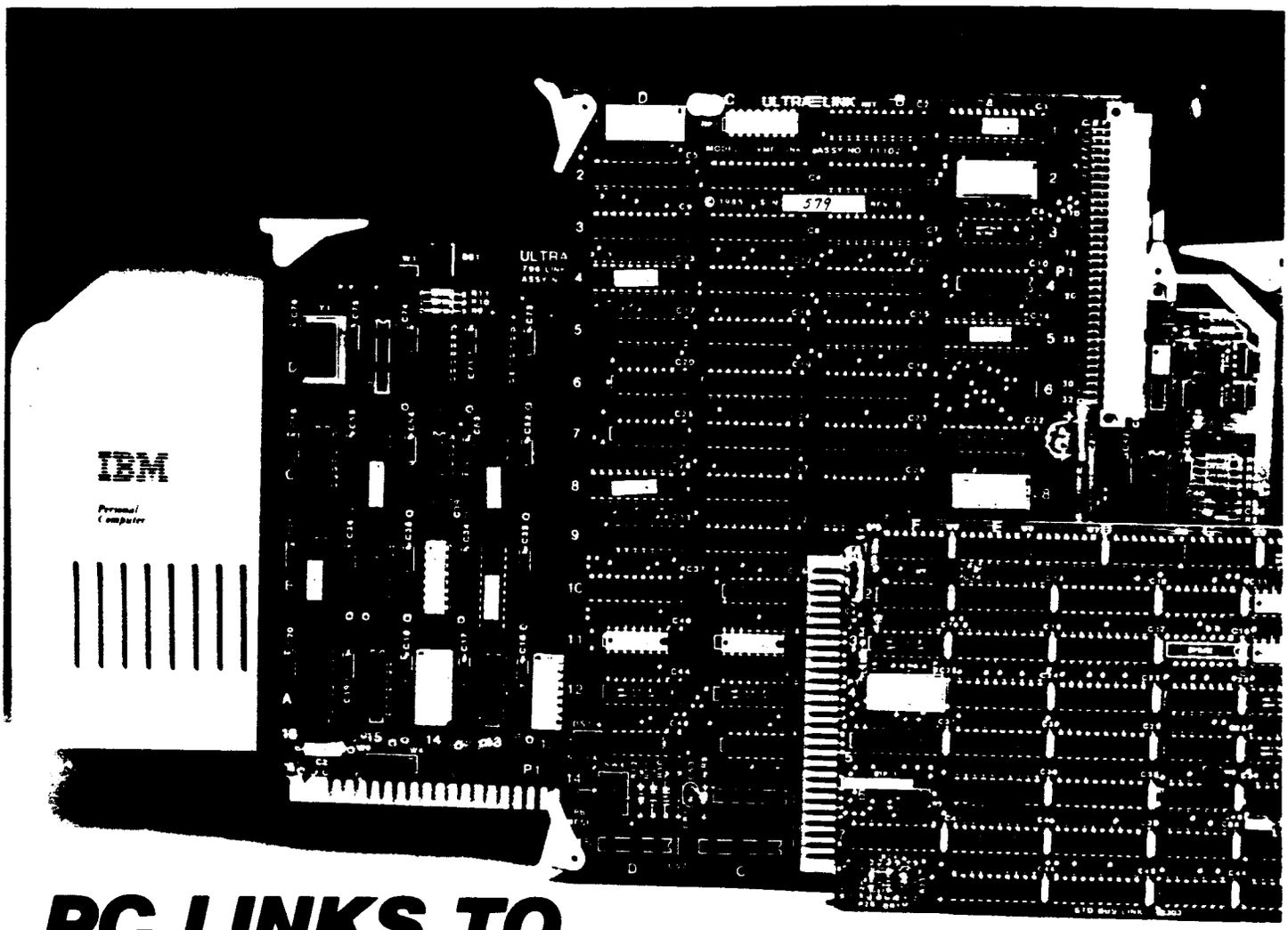
You then set up the Timer Control Register for that channel.

```
TIF1 - TIF0     (Doesn't matter, read only)
TIE1        = 0 (OFF, won't be using it)
TIE0        = 1 (ON, we want interrupts for this channel)
TOC1 - TOC0 = 0 (OFF, won't be using)
TDE1        = 0 (OFF, won't be using)
TDE0        = 1 (ON, we want downcounting)

00010001b = 11H = Value to output to TCR
```

Once the above value is output to the TCR, the downcounting will begin, and the interrupts will be generated at the specified intervals. It assumes that you have already set up your interrupt vectors and interrupt handling routine. The code in Listing 3gives an example of how to set up the PRTs and how to handle interrupts generated.

I hope that these two articles have given you some idea of the power and versatility of the Hitachi 64180, and perhaps helped you in utilizing some of its more advanced features. I did not cover some of its features, mainly the use of its serial ports, as they could not easily be covered in one or two articles, and you really have to have the Hitachi manual readily available to code for their usage. ■



PC LINKS TO VME, 796 AND STD BUS

Use your IBM PC for immediate access to the growing world of popular bus based products.

With the Link, the PC becomes a cost effective bus controller. The wide range PC support hardware and software make it a logical choice for expansion of present bus based systems or exploration of a target bus in new system designs.

The PC can operate important features of the target system bus directly for stand alone control or it can share the bus with other processors to become a powerful tool for bus development.

The PC can be used as a host or fault tolerant backup to bus CPU's in a multiprocessing environment. System-to-system interrupts are supported. The PC's DMA capability can be used for high speed intersystem data transfers.

Link operation is via direct parallel data path. Parity and optional optical isolation allow remote operation of systems at distances up to 50 feet. Any language or operating system can be used. The Link operates with the IBM PC, XT, AT, and compatibles.

Prices start at $845 for VME, $725 for 796 and $425 for STD Links. Delivery is from stock.

ULTRA-LINK INTERNATIONAL

1091 Airport Road
Minden, Nevada 89423
(702) 782-9758

PC, XT, AT are trademarks of IBM Corporation

ZSIG Corner

by Jay Sage, ZSIG Software Librarian

For this issue I will finally get to my long promised discussion of techniques for customizing the Z-COM automatic installation version of the Z-System. Before turning to the main subject, however, there are two other items I would like to cover. One is some follow-on discussion to the material I presented last time on recursive aliases; the other is a description of the latest ZSIG releases.

More on Recursive Aliases

Art Carlson forwarded to me a letter he received from Dreas Nielsen in response to my discussion of recursive alias techniques using VALIAS and ARUNZ. Dreas, well known in the ZCPR community for his excellent shell utility programs, GETVAR and RESOLVE, took me to task for doing something the hard (and not quite correct) way when there was a not-too-hard and completely correct way to do it. And he proved it by offering a marvelous technique for 'recursing' aliases without any of the problems I warned about with my technique. To me this is a wonderful example of the richness of ZCPR3. No matter how thoroughly one knows it, there are always significant new techniques and applications to be discovered. The learning and excitement never ends!

We will illustrate the technique using a simple example based on the following script that is designed to display a message on the screen showing how many times it has been run already. The script is:

```
alias ONELOOP:      ECHO THIS IS LOOP NUMBER
                    REG P6
```

As the ZCPR3 user register number 6 is incremented using the 'P'-for-plus option, its new value is displayed on the screen.

If we wanted to make the alias operate recursively, our first instinct, as described in my previous column, would be to expand it to the following:

```
alias MANYLOOP:     ECHO THIS IS LOOP NUMBER
                    REG P6
                    ECHO DO IT AGAIN?
                    IF IN
                    MANYLOOP
                    FI
```

As I explained last time, the trouble with this is that each time we run through the loop by answering the input prompt affirmatively we go one IF level deeper and accumulate another FI on the end of the command line. Eventually either the command line becomes too long or, more likely, we exceed the allowed eight levels of IF nesting.

Dreas's idea is based on turning this alias around so that there is no FI on the end to accumulate. He permutes it to the beginning of the script and writes the alias this way:

```
alias RECURSE2:     FI
                    ECHO THIS IS LOOP NUMBER
                    REG P6
                    ECHO DO IT AGAIN?
                    IF IN
                    RECURSE2
```

If this alias is invoked with a 'true' IF level already in effect, this will work just fine. The alias will drop one IF level lower at the FI command, do its main work, and return to the original IF level with the "IF IN" prompt. If the answer is affirmative, we will be back in the very same IF state we were in when the alias was invoked the first time. The alias will be re-invoked, and the whole operation will repeat. On the other hand, if the answer to the prompt is negative, the alias will stop running, and we will fall out the bottom at the same IF level as we entered originally but in the 'false' state.

To deal with the need to enter one IF level higher and to terminate the 'false' IF state at the end, we simply call the above alias from another alias with the following script:

```
alias RECURSE:      IF TRUE
                    RECURSE2
                    FI
```

The "IF TRUE" command will push us one IF level deeper and put us in a 'true' state. (This, by the way, is the first time that I have ever seen a use for the "TRUE" option with the IF command. If, like me, you never implemented that option because you could not figure out what on earth it could possibly be us for, you can substitute any other option that is guaranteec return a 'true' state, such as "IF NULL" with nothing after i "IF A=A"). The RECURSE2 alias will now run as we descri earlier, and when the recursion is terminated by a negative swer, the FI in RECURSE will terminate the resulting 'false' state and return us to the original IF level that we were at w RECURSE was invoked.

One has to stretch things a bit to come up with any disadvantages to this technique. The best I can do is the following two minor points. First, two aliases are required to get things started, and this will slow things down a bit, especially on a floppy-disk system. Secondly, the technique, while simple in principle, may not be easy to remember when one wants to set up a quick recursive alias, as in the application I described in my last article. The VALIAS/ARUNZ-type recursive alias may still be useful in such a case.

If you are a purist and have a strong, perhaps even irresistible, urge to do things right (and especially if you have a RAM disk so that you can afford to indulge such compulsions), Dreas's technique can be nicely automated by creating the following two slightly more complex and generalized aliases:

```
alias RECURSE       IF NULL $1
                    ECHO SYNTAX: $0 ALIASNAME PARAMETERS
                    ELSE
                    RECURSE2 $*
                    FI

alias RECURSE2      FI
                    $*
                    ECHO RUN "$1" AGAIN?
                    IF IN
                    $0 $*
```

Now suppose we have our edit/assemble/link alias as follows (where it, in turn, calls on the SYSLINK alias to perform the linkage with the libraries):

```
alias WORK          EDIT $1.Z80
                    Z80ASM $1/R
                    IF ~ERROR
                    SYSLINK $1
                    FI
```

If we enter the command "RECURSE" all by itself, we will get the message "SYNTAX: RECURSE ALIASNAME PARAMETERS." If we enter the command "RECURSE WORK MYPROG," the "IF NULL" test in RECURSE will be 'false' but will be toggled 'true' by the ELSE command. Then RECURSE2 will be invoked, and the command line and IF status will progress as shown in Figure 1. The "..." in the IF states designates any IF levels in effect at the time that RECURSE was called. Similarly, the command text 'pending' designates any command-line text following the invocation of RECURSE.

The figure does not show the operation of every command. The edit/assemble/syslink sequences are not shown in detail.

Figure 1

Step	IF state	Command Buffer Contents
1	...	RECURSE WORK MYPROG;pending
2	...	IF NULL WORK MYPROG;ECHO SYNTAX: RECURSE ALIASNAME PARAMETERS;ELSE;RECURSE2 WORK MYPROG;FI;pending
3	F...	ELSE;RECURSE2 WORK MYPROG;FI;pending
4	T...	RECURSE2 WORK MYPROG;FI;pending
5	T...	FI;WORK MYPROG;ECHO RUN "WORK" AGAIN?; IF IN;RECURSE2 WORK MYPROG;FI;pending
6	...	WORK MYPROG;ECHO RUN "WORK" AGAIN?;IF IN RECURSE2 WORK MYPROG;FI;pending
7	...	EDIT MYPROG.Z80;Z80ASM MYPROG/R;IF ~ERROR; SYSLINK MYPROG;FI;ECHO RUN "WORK" AGAIN?; IF IN;RECURSE2 WORK MYPROG;FI;pending
8	...	ECHO RUN "WORK" AGAIN?;IF IN;RECURSE2 WORK MYPROG;FI;pending
9	T...	RECURSE2 WORK MYPROG;FI;pending
10	...	ECHO RUN "WORK" AGAIN?;IF IN;RECURSE2 WORK MYPROG;FI;pending
11	F...	RECURSE2 WORK MYPROG;FI;pending
12	F...	FI;pending
13	...	pending

Evolution of the command line and IF states as the operation of the command "RECURSE WORK MYPROG" unfolds.

Between steps 8 and 9 we assume that the "IF IN" prompt was answered with a 'Y'. This brings us at step 9 back to the exact state we were in at step 4, and steps 4 through 8 will be repeated over and over so long as the "IF IN" prompt is answered affirmatively.

Between steps 10 and 11 we assume that the "IF ING"' prompt was answered with 'N', so we get the state shown in step 11. Since the current IF state is 'false', the command "RECURSE2 WORK MYPROG" is flushed, leaving us at step 12. The FI command terminates the 'false' IF and drops the flow state one level lower. Any pending commands that were on the command line after the invocation of RECURSE are now free to run at step 13.

I just tested out these aliases on my SB180 by putting them into my ALIAS.CMD file, which lives on the RAM disk. The scripts had to be altered slightly, with all invocations of RECURSE2 from within RECURSE and RECURSE2 being replaced by "ARUNZ RECURSE2 ..." or "ARUNZ $0 ...". They worked like an absolute charm. Thank you Dreas Nielsen!!!

A Z-Letters Feature

Thanking Dreas Nielsen for his letter reminds me that Art Carlson and I would like very much to start up a Z-Letters feature in TCJ, a kind of letters-to-the-editor section specializing in Z-System questions, comments, discussion, and ideas. I am willing to handle answering or otherwise responding to those letters, but I will depend on you readers to send them in. And rest assured that we are not looking only for letters like Dreas's containing brilliant suggestions. A good set of simpleminded questions would be quite well appreciated, thank you. They would help us a great deal in learning what aspects of Z-System are confusing to users so that we can try to clarify them.

SAGE MICROSYSTEMS EAST

Selling & Supporting
The Best in 8-Bit Software

- **Plu*Perfect Systems**
  - Backgrounder II: switch between two or three running tasks under CP/M ($75)
  - DateStamper: stamp your CP/M files with creation, modification, and access times ($49)
- **Echelon (Z-System Software)**
  - ZCOM: automatically installing full Z-System ($70 basic package, or $119 with all utilities on disk)
  - ZRDOS: enhanced disk operating system, automatic disk logging and backup ($59.50)
  - DSD: the incredible Dynamic Screen Debugger lets you really see programs run ($130)
- **SLR Systems (The Ultimate Assembly Language Tools)**
  - Assemblers: Z80ASM (Z80), SLR180 (HD64180), SLRMAC (8080), and SLR085 (8085)
  - Linker: SLRNK
  - Memory-based versions ($50)
  - Virtual memory versions ($195)
- **NightOwl (Advanced Telecommunications)**
  - MEX-Plus: automated modem operation ($60)
  - Terminal Emulators: VT100, TVI925, DG100 ($30)

Same-day shipping of most products with modem download and support available. Shipping and handling $4 per order. Specify format. Check, VISA, or MasterCard.

Sage Microsystems East
1435 Centre St., Newton, MA 02159
Voice: 617-965-3552 (9:00 a.m. - 11:15 p.m.)
Modem: 617-965-7259 (24 hr., 300/1200/2400 bps, password = DDT, on PC-Pursuit)

New ZSIG Releases

We have quite a few excellent new programs to release on ZSIG diskettes this month. I have not yet finalized exactly what will be included on release diskettes #2 and #3, but I don't want to pass up this opportunity to publicize them.

First of all, I am very happy to report that all the programs I proposed two issues back have now been written, and I will describe them first.

FCP10.LBR— This is the new flow control package (FCP) that I wrote with valuable assistance from Howard Goldstein (New Haven, CT). This program was described in some detail in the last column, so I will not say too much about it here. The two most important innovations are 1) the addition of AND and OR commands and 2) the FCP's ability to load and run IF.COM in high memory where it will not interfere with data in the beginning of the transient program area at 100H.

COMIF10.LBR— This is the companion transient IF processor that is intended to replace IF.COM. Also described last time, it offers an enormous number of new test conditions and syntax forms. This one was also written by me together with Howard Goldstein.

SETPATH1.LBR— This is an extended version of the original PATH command. It allows one optionally to add and remove path elements from the beginning or the end of the existing path. The path display is also improved. Robert Demrow, a fellow member of the Boston Computer Society CP/M Group wrote this one.

EDITND.LBR— Al Hawley (Z-Node #3 in Los Angeles) really went all out with tools to work with the named directory register (NDR). EDITND lets one edit the names directly in the NDR. Names and/or passwords can be added, deleted, and changed.

SAVNDR.LBR— After you've edited the NDR, this program lets you save the results to an NDR file. Again by Al Hawley.

LOADND12.LBR— The last in the Hawley set, this program can automatically update the names in the NDR using either the special file name system in LDSK or by loading an NDR file whose names are applied to the current floppy disk only.

The next ZSIG release will also include the following programs.

ZPATCH11.LBR— This is another masterpiece from Steve Cohen (Chicago, author of 'W', the wildcard shell, on the first ZSIG diskette). It is by far the best file patcher I have ever seen—a real joy to use. Steve sure is clever when it comes to shells! Who would have thought to make a file patcher into a shell? But this way one can run a command from inside ZPATCH and then return to one's exact place to continue working. The 'X' command in ZPATCH automatically runs the file one is currently patching (provided it is a COM file) so that one can see the effect of the changes. Marvelous!

MEX2Z.LBR— This is yet another brainstorm of NAOG chief Bruce Morgen (who has quite a stormy brain). MEX2Z and my adaptation of it for MEX-Plus, MEX+2Z, give the MEX communication programs the ability to 'shell', that is, to run a Z-System command apparently from within MEX. For example, if you enter the MEX command line 'CPM;CRUNCH FN.FT', you will exit from MEX, the file will be crunched, and then you go right back into MEX. This is especially handy when you are trying to debug a MEX script. Bruce picked up on the very clever trick introduced by Ted Emigh in FINDERR, namely, running a program that examines information left behind in memory by the program that ran before it. In this case it is the MEX command line buffer that is picked up. Even if you don't use MEX, it is worth looking at these programs just for their educational value.

FF10.LBR— I finally decided to do something about the shortcomings of FINDF, the utility for determining where files are located in your system. FF has a 16-bit configuration word that can be patched (roll out ZPATCH!) to define which drives should be searched by default (this is particularly useful when your constellation of drives has a hole in it, such as A, B, C, and F). You can override the default drive list by specifying a set of drives to scan on the command line. A whole list of file specifications can be given, and each one is automatically wildcarded, saving the user a lot of typing. If you want to find all programs starting with "SD", just enter "FF SD". The "SD" turns into "SD*.*" automatically. Similarly, "FF .LBR" will find all library files. "FF SD,.LBR" will find both.

PPIP15.LBR— This is the next step in the evolution of PPIP (PPIP14 is on ZSIG diskette #1). The main addition is support for DateStamper time and date stamping. Having fallen in love with DateStamper, I just had to have some file copying tools that would preserve the time and date information, so I added that capability to PPIP and to ZFILER.

ERRSET11.LBR— This little tool lets you either display the current or directly enter a new error handler command line. Strictly speaking, error handling in ZCPR3 is not performed by loading an error handling program but by executing an error handling command line. This command line is stored in a 16-byte string in the message buffer. When an error handler is installed by invoking its name manually from the command line, it writes only its name alone into that buffer. ERRSET lets you enter a complete error command line, such as A15:VERROR. By including an explicit DU: or DIR: form, the error handler will be found and loaded faster. On the other hand, ERRSET will let you enter the name of a nonexistent error handler, so watch out. Power has its price. Written by yours truly.

Customizing Z-COM

We now turn to the piece-de-resistance for this article—a discussion of techniques for customizing Echelon's automatically installing Z-System package known as Z-COM. This will be the first of a two-part series. This time I will cove the more elementary aspects of the subject, those modificatio that can be made by changing only data structures in the Z-CO files. Next time I will delve into customization techniques th involve serious hacking (such as modifying the Z-COM code self).

I will begin with an overview of what Z-COM is, t philosophy behind it, the procedure for installing it on a pa ticular computer, and how one uses it to create a Z-System automatically. Then I will give an elementary discussion of how it works and the structure of the COM file that magically transforms one's ordinary CP/M machine into a 'Z' machine. Next I will show you how to make some simple patches that eliminate the initialization operations that are performed by the startup file and make Z-COM come up ready to go instantly. This includes setting the wheel byte, defining the symbolic command search path, putting in the terminal capability descriptor (TCAP) for the user's terminal, installing the user's named directories, installing an external error handler, and even setting up an initial shell.

With those techniques mastered, we can then go on to make changes to the system modules. The simplest of these is replacing the standard ZRDOS that comes with Z-COM with the

latest-and-greatest Public-ZRDOS-Plus version 1.7. Slightly more complex is the replacement of the environment descriptor (ENV) and the two command modules: the RCP (resident command package) and the FCP (flow control package). For these changes we have to edit some configuration files, assemble new code modules, and install the new modules into the Z-COM file. By this point you will be ready to generate and install a new version of the ZCPR3 command processor, an operation that requires one important extra step (a simple one, but one that must not be overlooked).

Why Z-COM

Consider this situation. The Z-System is a wonderful replacement for CP/M, one that greatly enhances the utility and ease of operation of an 8-bit Z80-compatible computer. I can't understand why anyone would not enjoy and benefit from its features and capabilities. However, installing it in the conventional fashion required changing the BIOS (Basic Input Output System), the hardware-dependent part of the operating system. Unfortunately, many (actually most) manufacturers consider their BIOS to be proprietary (or embarrassingly poorly written), and they refuse to release the source code. And even if they do make it available, perhaps for an extra fee, what if you do not know how to make the required changes to support the ZCPR3 command processor? Well, enter Z-COM.

Z-COM was the brilliant conception of Joe Wright, the nation's preeminent BIOS writer (author of the BIOSes for the Ampro Little Board, Micromint SB180, and the soon-to-be-available ON! computer from Oneac). With Z-COM one does not need the BIOS source code because there are no changes to make in it. One doesn't even need MOVCPM. Z-COM runs on almost any standard CP/M 2.2 system, converting itin situ to a Z-System. There are a few CP/M 2.2 computers on which Z-COM will not work (usually because at least some part of their memory space operates in a funny way), but the great majority will. For those of you with CP/M version 3 (aka CP/M-Plus—a real misnomer in my opinion), I'm afraid you are out of luck. CP/M 3 is fundamentally different from CP/M 2.2, and no one has yet been able to concoct magic powerful enough to transform it into a Z-System.

I hope my discussion here will inspire some of you to purchase Z-COM. If it does, then see the ads in TCJ by Sage Microsystems East and Echelon for more information. I personally think that a modified Z-COM is the best way to implement Z-System, because, as we will see by the end of this series of articles, it gives one far greater flexibility than with a manually installed system.

Installing Z-COM

Here is a brief description of how one gets Z-COM running on one's system. The standard procedure goes like this. Take a freshly formatted disk, 'sysgen' the CP/M 2.2 operating system to it, copy onto it all the files on the Z-COM release disk, put this disk into drive A, and reboot the system either with the reset button or by entering control-c. Now enter the command "SUB ZCCOM". This starts a batch operation, and you can now sit back, relax, and watch your computer do all the work. Toward the end of the process, which takes a couple of minutes, a program called TCSELECT will run and ask you to choose your terminal from a large menu. The result will be a file called MYTERM.Z3T containing information about your terminal that permits Z-System programs to perform screen operations without installation.

What's Going On During Installation

In the first part of the installation process, the Z-COM system creation program ZCLD.COM determines the memory address at which your BIOS operates and generates a corresponding Z-System. To do this, it reads in a number of special relocatable files (most with file type SPR, which stands for system page relocatable) and produces four new files: ZC.COM, ZCX.COM, ZC.ENV, and ZC.CP. The first file, whose operation we will describe in detail below, is the one that transforms the vanilla CP/M system to the amaretto-double-chocolate almond Z-System with jimmies (if you don't know what jimmies are, ask someone from Boston).

The second file, ZCX.COM, is a program that transforms you back. Why, you ask, would one ever go back to vanilla after experiencing the ecstasy of amaretto-double-chocolate-almond? Well, Z-System does have one significant drawback. You don't get all those great features for free. They cost a considerable chunk of TPA (transient program area)—5.5K in the case of Z-COM (though we will see next time how to reduce this if you are willing to forego some of the features). The smaller TPA has almost never caused any problem with the programs I use, but some people do have problems, and it is nice to know that a simple "ZCX" command will bring back the old CP/M with its full-sized TPA. Next time I will explain how we can even change to a different Z-COM system with a larger TPA.

The third file, ZC.ENV, is the environment descriptor file for the system that ZCLD created. It is automatically included in ZC.COM, so it does not have to be loaded using LDR, but it is used by Z3INS to install the environment information into the utilities.

The last file, ZC.CP, does not even appear in a directory listing; it is hidden up in user area 15, out of harm's way. The user is not normally concerned with this file, though if you want to create another Z-COM system disk, you have to remember to copy it, as well as the others mentioned above, to the new diskette. We will discuss its purpose later.

How ZC.COM Works

As we said above, ZC.COM is the program that transforms your mundane CP/M system into an exciting and powerful Z-System. How does it do it? Several simple principles are involved.

ZC.COM is basically a loader program. The file itself consists of two parts. The first page of code (100H to 1FFH) is the loader code. The second part (200H to 2DFFH) is the memory image of a Z-System that is copied into place by the loader. The process is like the 'big bang' theory of creation—the whole Z-System just appears complete in one operation!

The memory map of the ZCOM-System generated for my BigBoard I computer, on which I performed these experiments, is shown in Figure 2. Its real CP/M BIOS is at E800H. The Z-System addresses were determined by running the utility SHOW.COM after Z-COM was loaded. The corresponding addresses in the ZC.COM file were obtained by inspecting it with a debugger. Once a few addresses (like the RCP and FCP, which have obvious headers), were determined, the rest was obvious. The ZC.COM system image is at a constant offset from the real system. In this example, that offset is BA00H. If Z-COM is installed on a different system, the real system addresses and the offset value will be different, but the addresses of the system segments in the ZC.COM image will be the same. In general, the offset between the corresponding addresses will be 2E00H less that the address of the native BIOS.

Figure 2

System Component	ZC.COM Address	System Address
CPR	0200 - 09FF	BC00 - C3FF
ZRDOS	0A00 - 17FF	C400 - D1FF
Virtual BIOS	1800 - 19FF	D200 - D3FF
Named Directory Register	1A00 - 1AFF	D400 - D4FF
Shell Stack	1B00 - 1B7F	D500 - D57F
Z3 Message Buffer	1B80 - 1BCF	D580 - D5CF
External FCB	1BD0 - 1BF3	D5D0 - D5F3
PATH	1BF4 - 1BFE	D5F4 - D5FE
Wheel Byte	1BFF - 1BFF	D5FF - D5FF
Environment Descriptor	1C00 - 1C7F	D600 - D67F
TCAP	1C80 - 1CFF	D680 - D6FF
Multiple Command Line	1D00 - 1DCF	D700 - D7CF
External Stack	1DD0 - 1DFF	D7D0 - D7FF
Resident Command Package	1E00 - 25FF	D800 - DFFF
Flow Control Package	2600 - 27FF	E000 - E1FF
I/O Package	2800 - 2DFF	E200 - E7FF

Addresses of system components in the ZC.COM file and in the example system for which it was generated.

How does this system function? If you are familiar with Z systems, you probably recognize all of the system components above except for the one called 'virtual BIOS'. That is where the key to Z-COM lies. Remember, we needed a BIOS that would run below the Z buffers, but we had no way to relocate the actual BIOS. So instead we create a virtual BIOS—a block of code structured just like a real BIOS. It has a table of jump instructions, one after the other, that perform the required BIOS functions: CBOOT, WBOOT, CONST, CONIN, CONOUT, LIST, and so on. How does this virtual BIOS actually carry out those functions without knowing anything about the system hardware? Easy! It simply jumps to the corresponding entry points in the real BIOS!

Well, it actually is not quite that easy. There are a few special details that have to be taken care of. Most of the functions are performed as described above, but there are some important exceptions. The most important one is the WBOOT, or warm boot, function. Normally when a warm boot is performed, the CPR (and often the BDOS as well) is reloaded from the system tracks of the A diskette. If that were allowed to happen here, goodbye Z-System! ZC.COM would only work until the first warm boot occurred, and then we would have to run it again. Not very satisfactory!

To prevent that from happening and to keep Z-COM running, the virtual BIOS 'traps' warm boot calls. That is a fancy way of saying that instead of simply passing the call to the real BIOS it does the work itself. What does it do? Well, it has to reload the ZCPR3 command processor to the proper address, BC00H in this example. Since the ZCPR command processor does not reside on the system tracks, we have to get it from somewhere else. Joe Wright could have gotten it from records 2 through 17 in ZC.COM, but he chose instead to maintain a separate file with just the image of the command processor. Remember the file ZC.CP that we mentioned earlier, the one stashed away in A15:? That's it.

Although I don't know of any reason why CBOOT, the cold boot routine, would ever be called once the computer was initially booted up, the CBOOT routine is also trapped by the virtual BIOS and vectored (another fancy word) to the same code as the virtual warm boot.

In Echelon's simpler auto-install package Z3-DOT-COM, which does not have support for Input/Output Packages (IOPs), the story would now be complete. The IOP, however, has to get first shot at some of the BIOS I/O routines, namely console status, console input, console output, list output, list status, punch output, and reader input. In a manual installation of Z-System with IOP support, the BIOS code would have to be modified. With Z-COM this is quite straightforward. The virtual BIOS calls to these functions simply go (are vectored) to the appropriate entry points in the IOP module. The initial IOP code included in ZC.COM is just a dummy IOP that simply turns around and forwards the calls from the IOP to the real BIOS entry points.

• Z Best Sellers •

Z80 Turbo Modula-2 (1 disk) $69.95
The best high-level language development system for your Z80-compatible computer. Created by Borland International. High performance, with many advanced features, includes editor, compiler, linker, 552 page manual, and more.

Z-COM (5 disks) $119.00
Easy auto-installation complete Z-System for virtually any Z80 computer presently running CP/M 2.2. In minutes you can be running ZCPR3 and ZRDOS on your machine, enjoying the vast benefits. Includes 80+ utility programs and **ZCPR3: The Manual**.

Z-Tools (4 disks) $150.00
A bundle of software tools individually priced at $260 total. Includes the ZAS Macro Assembler, ZDM debuggers, REVAS4 disassembler, and ITOZ,ZTOI source code converters. HD64180 support.

PUBLIC ZRDOS (1 disk) $59.50
If you have acquired ZCPR3 for your Z80-compatible system and want to upgrade to full Z-System, all you need is ZRDOS. ZRDOS features elimination of control-C after disk change, public directories, faster execution than CP/M, archive status for easy backup, and more!

DSD (1 disk) $129.95
The premier debugger for your 8080, Z80, or HD64180 systems. Full screen, with windows for RAM, code listing, registers, and stack. We feature ZCPR3 versions of this professional debugger.

Quick Task (3 disks) $249.00
Z80/HD64180 multitasking realtime executive for embedded computer applications. Full source code, no run time fees, site license for development. Comparable to systems from $2000 to $40,000! Request our free Q-T Demonstration Program.

Z-System OEM inquiries invited.
Visa/Mastercard accepted. Add $4.00 shipping/handling in North America; actual cost elsewhere. Specify disk format.

Echelon, Inc.
885 N. San Antonio Road • Los Altos, CA 94022
415/948-3820 (Order line and tech support) Telex 4931646

Easy Z-COM Patches

Now that we understand what Z-COM is and how it work we can start to make some changes. If you look at the STF startup alias with Z-COM, you will see that it turns on the whe byte, loads the TCAP and named directory register, and sets t the symbolic search path. For our first set of patches, we w eliminate the need for a startup alias. We will make the system come up fully tailored to our preferences, and we will save the time wasted by the STRT alias.

The first step is to get Z-COM running and to use the utility programs to set it up as we like it. We will generally turn the wheel byte on (the STRT alias presumably already ran WHEEL to do that for us). We can set up our named directories using MKDIR and LDR or the new ZSIG named directory editor EDITND. We can choose our path using the PATH command or the new ZSIG SETPATH utility. An external error handler can be defined in the message buffer by invoking the desired error handler manually at the command line or by running the ERRSET utility (this can include a DU or DIR specifier to speed up the search for the error handler). Finally, if we want to, we can even invoke a shell, such as the history shell HSH.

Now all we have to do is clone this system using our favorite debugger. If you can afford Echelon's DSD, I cannot recommend it highly enough. It will quickly spoil you. On the other hand, I will describe here the procedure using Prof. Falconer's lovely DDTZ, a public-domain Zilog-mnemonic version of DDT. Here is the sequence of commands to use. Don't forget that the addresses that refer to the real system are the ones for my BigBoard. You should make a table like that in Fig. 2 with the addresses for your system and make the appropriate substitutions in the commands below.

```
A0:SYS>ddtz zc.com          ; Run debugger and load ZC.COM
-md400,d6ff,1a00            ; Copy running NDR, Shell Stack, MSG
                            ; ..buffer, PATH, WHL, ENV, and TCAP
                            ; ..into ZC.COM image
-g0                         ; Exit from debugger
A0:SYS>save 2dh zcnew.com   ; Save new version of ZC
```

If you have a shell running while this process is being carried out, you have to include the SAVE command on the same line as the DDTZ command, since when the shell loads it wipes out the memory image in the TPA. By putting the SAVE command on the same line, it is run before the shell is reloaded.

I have used the above technique and found it to work very nicely, but a word of caution is in order. A purist would copy the memory areas only for the specific segments to be modified. The external file control block and the parts of the message buffer other than the byte at offset 0 (D580, where the error-handler-flag is kept) and bytes at offsets 10H to 1FH (D590-D59F, where the error handler command line is kept) would not be copied. Copying the entire message buffer works as described above, but if you try to make a ZEX batch file to do this, you will get into trouble, since ZCNEW.COM will then contain an image of the message buffer with the ZEX-running flag set.

To test ZCNEW, make a new STRT.COM alias that just echoes a signon message (first rename the old one in case you want it back later), run ZCX to exit from Z-COM, and then run ZCNEW. Your own personalized Z-System should pop (almost) instantly to life. After determining that it is really working correctly, you can rename ZCNEW.COM to ZC.COM. Again, I recommend keeping old versions with names like ZC1.COM, ZC2.COM, and so on, on an archive disk.

Putting in a New DOS

Now let's make a change that could not be accomplished using utility programs, as all of the above changes could be. Let's replace the older, non-public version of ZRDOS that comes with Z-COM with the latest version 1.7 of Public-ZRDOS-Plus. The first part of the process is same as that in a manually installed Z-System. One takes the ZRDOS generating program ZRDINS17.COM, installs it using Z3INS, and runs it. Having been installed, it will automatically know the three facts it needs: where the DOS, ENV, and wheel byte are located. After it finishes running, there will be a binary image file called ZRDOS17.BIN.

With a manually installed Z-System we would now have to go through a somewhat complex process involving 'sysgening' a system image from the system tracks, patching in the new DOS using a debugger, and 'sysgening' the image back onto the system tracks. With Z-COM things are actually considerably easier, since the two 'sysgening' steps can be skipped. Here is the command sequence:

```
A0:SYS>ddtz zc.com          ; Run debugger and load ZC.COM
-izrdos17.bin               ; Initialize the file control block
-r900                       ; Read with offset 900h so that the object
                            ; ..file will load at A00h
-g0                         ; Warm boot out of debugger
A0:SYS>save 2dh zcnew.com   ; Save new version
```

As before, exit from Z-System with the ZCX command (or just hit the reset button if that is more convenient). Then load the new system and run the DOSVER utility. Voila! There is version 1.7.

Now let me show you a trick that makes patching even easier and doesn't require a debugger at all. Just use the following command sequence:

```
A0:SYS>get 100 zc.com       ; Load ZC.COM at address 100H
A0:SYS>get a00 zrdos17.bin  ; Load ZRDOS17.BIN at address A00H
A0:SYS>save 2dh zcnew.com   ; Save the new memory image
```

With this technique there is no computing of offsets, and it uses only ZCPR3 built-in commands (GET and SAVE). The one thing you have to remember is that the GET command loads entire files, and so this technique can only be used to patch in things that come in contiguous blocks with a length that is an integral multiple of 128 bytes. If you look at the memory table of ZC.COM, however, you will see that this includes all system modules that ever exist in the form of files: the CPR, DOS, NDR, ENV, TCAP, RCP, FCP, and IOP. These patches can be done very nicely from alias scripts. For example, since one might often change one's named directories, the following alias might be handy:

```
alias PUTNDR:

        IF NULL $1
        ECHO SYNTAX: $0 NDR-FILE-NAME
        ELSE
        ERA ZCNEW.COM
        GET 100 ZC.COM
        GET 1A00 $1.NDR
        SAVE 1DH ZCNEW.COM
        ECHO TEST ZCNEW.COM
        FI
```

Similar aliases could be used for replacing other modules such as RCPs and FCPs.

New Command and System Modules

Speaking of RCPs and FCPs, let's talk a little about how we generate new versions of these and other system modules. There is no need to discuss the patching techniques. They are the same as what we have already seen. The question is: how do we generate the new SYS.RCP, SYS.FCP, and SYS.ENV modules?

The procedure is the same as in a manual install system, except that in a manual install system we already had the necessary Z3BASE.LIB file that is required for the assembly of new system modules. So far we have not needed one with Z-COM. To make things easier, Joe Wright has kindly provided us with a generalized Z3BASE.LIB file. All we have to do to adapt it to our system is to use an editor to fill in the address of the environment descriptor in the "Z3ENV SET ..." definition near the top of the file. One further hint: while you are at it, change the SET to an EQU. If you use Z3BASE.LIB to assemble a file in Zilog mnemonics, the assembler will not accept SET, since that is a Zilog opcode. You would have to change it to ASET in that case, but there is no reason not to use the universal EQU. Some

programs also require definitions for YES and NO, so you might want to add the lines:

```
YES     EQU     TRUE
NO      EQU     FALSE
```

One last addition. The RCP, FCP, and ENV modules do not need it, but other files, and most notably the command processor code, require an equate to define the address of the entry point to the command processor. This is a part of the normal Z3BASE file, but Joe Wright forgot to put it in. Someplace after the Z3ENV equate, add the line:

```
CCP     EQU     Z3ENV - 1A00H
```

Now you should be all set to assemble up new system modules, like the new FCP10 from ZSIG. You might also want to assemble a customized SYS.ENV with the correct values of maxdrive and maxuser and your own choice of printer and console definitions (see "ZCPR3, The Manual" for details).

"ZCPR3, The Manual" explains in detail how to carry out the assemblies, though I find that the discussion there makes the process look more complicated than it really is. Basically, you assemble the module to a HEX file, convert the HEX file to a COM file using MLOAD, and then rename the COM file to the proper name for the module (e.g., "REN SYS.FCP=F-CP10.COM"). If you have the SLR assemblers, you can assemble the module directly to a COM in a single pass and skip the MLOAD step. Finally you test the module by loading it using LDR.

Replacing the Command Processor

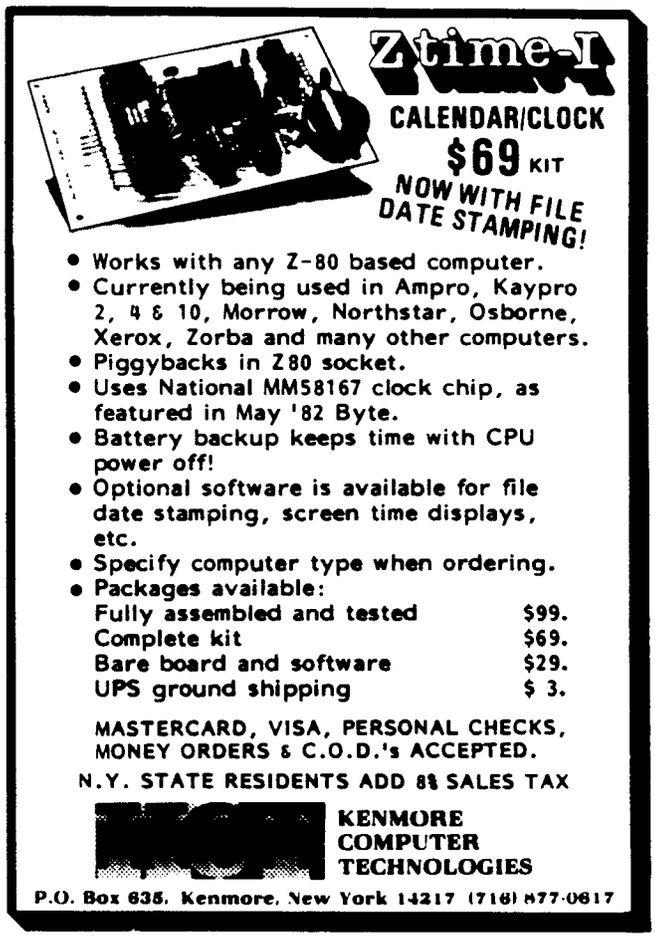
Replacing the command processor is no more difficult than changing any of the other modules in the system. In fact, as we will soon see, it is much easier to test a new CPR with Z-COM than it is with the manually installed Z-System.

The basic procedure is as we described for the RCP or FCP. As usual, edit the configuration LIB file (Z3HDR.LIB or Z31H-DR.LIB if you are using my experimental version ZCPR315D) and assemble the code. One additional hint. Again, if you have the fabulous SLR Systems assemblers SLRMAC, Z80ASM, or SLR180, select the option to assemble directly to a COM file, not to a HEX file. It is a shame that so many books and articles have taught the unnecessarily complex patching method based on HEX files with their extremely tricky offset calculation. It is much easier to work with a COM file where the load offset in the debugger is quite straightforward, namely, 100H below where you want the file to go, no matter what address it was assembled for. If you are using an assembler that can produce only a HEX file, then use the public-domain MLOAD to convert it to a COM file just as was done to make the RCP and FCP files.

Now that you have ZCPR3.COM or some such file, all you have to do is substitute it for the file ZC.CP in A15:. It would be a sign of foolhardy optimism to destroy the original ZC.CP. I would recommend renaming it to something like ZCOLD.CP and then copying the new one into place with a command like "PPIP A15:ZC.CP=ZCPR3.COM". Now just hit control-c to warm boot, and the new CPR will be running!

If things did not work out as you intended (and especially if

AMPRO
COMPUTERS INCORPORATED
AUTHORIZED DEALER
BEAR ELECTRONICS
SAVE 10% on AMPRO
ALL AMPRO PRODUCTS READY FOR IMMEDIATE SHIPMENT
SOME EXAMPLES: (Other AMPRO products available at similar savings.)
MODEL | DESCRIPTION | REG. PRICE | DISCOUNT PRICE
1B-1 | Little Board SBC | 249.00 | 224.10
1B-2 | Little Board/PLUS SBC | 289.00 | 260.10
2A-2 | Little Board/186 SBC | 495.00 | 445.50
2A-P | Proto Board | 179.00 | 161.10
2VR | Video RAM Emulator | 195.00 | 175.50
3A-1 | STD Bus SCSI I/O Board | 119.00 | 107.10
122A | CP/M System w/2 Drives | 995.00 | 895.50
222 | PC-DOS System w/2 Drives | 1295.00 | 1165.50
321 | Expandable PC-DOS System | 1395.00 | 1255.50
Please add $3.00 ($8.00 on systems) for UPS shipping & handling.
California residents add appropriate sales tax.
VISA
CHECKS
PO's
Some restrictions apply to credit and large quantity orders. Call for information. Allow 2 - 3 weeks for 122A, 222 or 321 systems.
CALL NOW - CALL COLLECT (415) 376-0125
BEAR ELECTRONICS, P.O. Box 61, Moraga, CA 94556


Ztime-I
CALENDAR/CLOCK
$69 KIT
NOW WITH FILE DATE STAMPING!
• Works with any Z-80 based computer.
• Currently being used in Ampro, Kaypro 2, 4 & 10, Morrow, Northstar, Osborne, Xerox, Zorba and many other computers.
• Piggybacks in Z80 socket.
• Uses National MM58167 clock chip, as featured in May '82 Byte.
• Battery backup keeps time with CPU power off!
• Optional software is available for file date stamping, screen time displays, etc.
• Specify computer type when ordering.
• Packages available:
Fully assembled and tested $99.
Complete kit $69.
Bare board and software $29.
UPS ground shipping $ 3.
MASTERCARD, VISA, PERSONAL CHECKS, MONEY ORDERS & C.O.D.'s ACCEPTED.
N.Y. STATE RESIDENTS ADD 8% SALES TAX
KENMORE COMPUTER TECHNOLOGIES
P.O. Box 635, Kenmore, New York 14217 (716) 877-0617

the new CPR just crashed the system), you can reboot and reload ZC COM. Remember, you did not change the CPR image in ZC.COM yet, and things will be fine so long as you don't warm boot. Before that happens, you should rename ZCOLD.CP back to ZC.CP. Then you can try again. That is sure a lot easier than all the 'sysgening' required to test a new CPR with the manually installed Z-System.

If all went well, the system is now running just the way you hoped it would. Once you've tested it and are satisfied that it really is working correctly, you can patch the new CPR into ZC.COM so that it will be ready immediately after Z-COM loads. The following commands will do it:

```
A0:SYS>get 100 zc.com
A0:SYS>get 200 a15:zc.cp
A0:SYS>save 2dh zcnew.com
```

As usual, once you have verified that ZCNEW is working, rename it to ZC.COM.

Summary and Plans

That wraps things up for this time. You now know how to make enough patches to mold a Z-COM system pretty well to your own tastes. If you have Z-COM, I hope you will experiment some before the next column appears. In that column we will really dig into Z-COM and do some wild things. If you have any suggestions or questions, please send them along. My address and phone numbers are listed below. As a little reward for those of you who plowed through this whole piece (or who were taught the trick I was as a freshman in college to read the end of a paper or essay before you start at the beginning), here is a little tidbit. ZCPR version 3.3 is coming very soon. I already have a nearly complete version of it patched into the Z-COM I worked up for this article. I expect that Z33 will have been released or will be very close to release by the time this column appears, and I will devote some space to a discussion of its new features in the next column. ■

Jay Sage
1435 Centre Street
Newton Centre, MA 02159
voice: 617-965-3552
modem: 617-965-7259

Z — Letters

As Jay Sage mentioned in his column, we are starting a Z-Letters section for your comments and questions, and the following letter from Dreas Nielsen is in response to TCJ#27.

Aliases and Recursion

Jay Sage's discussion of flow control in nested and recursive aliases (TCJ#27) overlooks a simple technique that is, I think, better than the one that he presents. Use of XIF and the special aliases generated by VALIAS involves two drawbacks. For example, consider the multiple-command line;

```
IF <condition>
<recursive alias>
ELSE
<other action>
IF
<more commands>
```

If the alias uses XIF (but is not recursive in the VALIAS style) the "other action" will always be executed, even when the condition is true. If it is a VALIAS-style recursive alias, all commands following it will be discarded.

Recursive aliases can easily be constructed that have neither of these disadvantages. An example will serve better than an expalanation:

```
Alias RECURSE:
    IF A=A
    RECURSE2
    FI

Alias RECURSE2:
    FI
    ECHO THIS IS LOOP NUMBER
    REG P6
    ECHO DO IT AGAIN?
    IF IN
    RECURSE2
```

Neither of these need be the special "recursive" aliases generated by VAILAS. As you can see, the truly recursive alias (RECURSE2 in this example) should start out by popping one IF level, and you must simply ensure that you push one TRUE IF level before calling it. (Here the IF <str>=<str> test is used; the IF TRUE command could be used if you system includes it.)

Without wishing to offend Jay Sage, who is contributing so much to the Z community, I'd like to point out that by avoiding the "recursive" aliases generated by VALIAS, you can nest aliases freely without worrying that you may have created one of them in "recursive" mode, and that it will blitz the rest of your command line.

Dreas Nielsen

Using SCSI for Real Time Control

Separating the Memory and I/O Buses

by Rick Lehrbaum, Vice President Engineering, Ampro

Introduction

The popular microcomputer system buses (Multibus, VME, STD, S-100, and IBM PC Bus) have evolved to fit the needs of system designers in a wide variety of applications. Each combines the capabilities of high speed system memory expansion with the ability to access and expand system input/output ports and devices.

In recent years, microprocessors have become more powerful both in terms of raw clock speed and in memory access efficiency. Furthermore, with the continual exponential increase of memory device density, the price per bit has plummeted, making system memory a highly expandable system resource. Consequently, much of the evolution of the popular microprocessor backplane buses has focused on enhancing the ease, flexibility, and efficiency of the system memory expansion.

But memory expansion is obviously not the only purpose of the microcomputer system's backplane bus. The system backplane bus must also be able to accomodate a wide variety of input/output (I/O) interfaces to devices such as consoles, printers, disk drives, communication interfaces, and more. Support for the systems's I/O device interfacing and control is therefore the second key requirement of the system's backplane bus.

These two functions—memory and I/O expansion—have distinct, and in many ways conflicting, requirements. Speed is often the main consideration in the design of memory expansion interface buses. In a high performance system the backplane bus must be optimized for the microprocessor's memory bus interface signals and timing. Maximum memory access speed is achieved by a variety of techniques, including both asynchronous and synchronous designs, CPU pipelining, memory prefetch, etc., all of which are coupled tightly with the particular microprocessor architecture (8086, 68000, 16032, etc.) on which the system CPU is based.

I/O devices, on the other hand, are accessed relatively infrequently in comparison with memory devices, and have much slower inherent response times. Access time is therefore generally not as critical in design of I/O device interfaces as with memory. Instead, other factors such as data buffering, interrupt response, direct memory access (DMA) efficiency, and I/O control protocols, are more significant.

The typical system backplane bus is thus called upon to provide both high speed memory expansion with its inherent speed requirements, and efficient I/O interface expansion with its real time, asynchronous requirements. The result is a bus that is a compromise between the requirements of high speed memory access, which needs to be finely tuned to the microprocessor being used, and that of I/O device interfaces, which are less stringent and therefore capable of a greater degree of generalization without significant loss of system efficiency.

The first generation of microcomputer backplane buses (S-100, STD, and Multibus) all supported both memory and I/O expansion within a single set of signals, and suffered from the limitations of the required compromise.

More recently, some second generation microprocessor buses such as Multibus II and VME have begun to provide multiple interconnects which improve the coexistance of high speed memory expansion and efficient and flexible I/O expansion on a single backplane bus.

A significant disadvantage of these second generation buses' multiple interconnect scheme is the lack of standardization between the respective buses. In the case of memory expansion, this is probably justified by the requirements to provide minimum access times and the coupling of that objective with the specific microprocessor's chip-level signals.

There remains, however, an opportunity to simplify the problem of system integration with many types of I/O device interfaces by standardizing on a universal small system I/O bus. Such a bus has been in development over the past seven years, and is now a widely accepted and fully specified standard.

SCSI as a Universal I/O Bus

The Small Computer System Interface (SCSI) was originally developed by Shugart Associates to provide a standardized interface between computer systems and intelligent disk controllers. SCSI is now an accepted US standard (ANSC X3T9.2) in wide use by computer board and system manufacturers.

In the introduction to the SCSI specification, the principle goal of SCSI is stated to be:

> "...to facilitate the integration of small computers and intelligent peripheral devices, particularly storage devices..."

Although SCSI was originally developed for use in mass storage applications (i.e. disk and tape), there is nothing in its design which limits it to that realm. On the contrary: the Small Computer System Interface is, as its name implies, a fully general I/O interface which can be used for the interconnection among all types of small computer system devices.

As a general purpose I/O expansion bus, SCSI has the following featues:

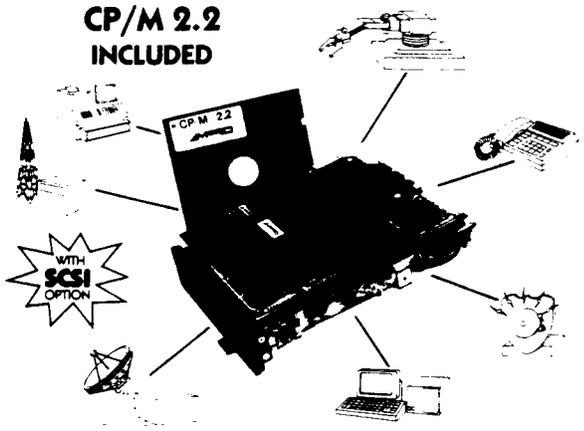
Universal Bus Timing—SCSI is a buffered, asynchronous I/O bus with request/acknowledge handshake signals. CPU-related timings do not affect bus activity, and dissimilar processors can share common peripherals. The SCSI bus therefore effectively transcends the characteristics of the host computer(s).

Peripheral Sharing—SCSI's bus arbitration scheme allows several bus hosts to share a single SCSI bus along with all connected peripherals.

Little Board™ $249

The World's Least Expensive CP/M Engine

CP/M 2.2 INCLUDED



- 4 MHz Z80A CPU, 64K RAM, Z80A CTC, 4-32K EPROM
- Mini/Micro Floppy Controller (1-4 Drives, Single/Double Density, 1-2 sided 40/80 track)
- 2 RS232C Serial Ports (75-9600 baud & 75-38, 400 baud), 1 Centronics Printer Port
- Power Requirement: +5VDC at .75A; +12VDC at .05A / On board -12V converter
- Only 5.75 x 7.75 inches, mounts directly to a 5-1/4" disk drive
- Comprehensive Software Included:
  - Enhanced CP/M 2.2 operating system with ZCPR3
  - Read/write/format dozens of floppy formats (IBM PC-DOS, KAYPRO, OSBORNE, MORROW...)
  - Menu-based system customization
  - Operator-friendly MENU shell
- OPTIONS:
  - Source Code
  - TurboDOS
  - ZRDOS
  - Hard disk expansion to 60 megabytes
  - SCSI/PLUS™ multi-master I/O expansion bus
  - Local Area Network
  - STD Bus Adapter

BOOKSHELF™ *Series 100*

Fast, Compact, High Quality, Easy-to-use CP/M System

Priced from $895.00
10MB System Only $1645.00

- Ready-to-use professional CP/M computer system
- Works with any RS232C ASCII terminal (not included)
- Network available
- Compact 7.3 x 6.5 x 10.5 inches, 12.5 pounds, all-metal construction
- Powerful and Versatile:
  - Based on Little Board single-board computer
  - One or two 400 or 800 KB floppy drives
  - 10-MB internal hard disk drive option
- Comprehensive Software Included:
  - Enhanced CP/M operating system with ZCPR3
  - Word processing, spreadsheet, relational database, spelling checker, and data encrypt/decrypt (T/MAKER III™)
  - Operator-friendly shells, Menu, Friendly™
  - Read/write and format dozens of floppy formats (IBM PC-DOS, KAYPRO, OSBORNE, MORROW...)
  - Menu-based system customization

DISTRIBUTORS

ARGENTINA: FACTORIAL, S.A., (1) 41-0018, TLX 22408 **BELGIUM:** CENTRE ELECTRONIQUE LEMPEREUR, (041) 23-4541, TLX 42621 **CANADA:** DYNACOMP COMPUTER SYSTEMS LTD., (604) 872-7737 **ENGLAND:** QUANT SYSTEMS, (01) 253-8423, TLX 946240 REF 19003131 **FRANCE:** EGAL+, (1) 509-1800, TLX 620893 **SPAIN:** XENIOS INFORMATICA, 593-0822, TLX 50364 **AUSTRALIA:** ASP MICROCOMPUTERS, (613) 500-0628 **BRAZIL:** CNC-DATA LEADER LTDA., (41) 262-2262, TLX 041-6364 **DENMARK:** DANBIT, (03) 66-20-20, TLX 43558 **FINLAND:** SYMMETRIC OY, (0) 585-322, TLX 121394 **ISRAEL:** ALPHA TERMINALS, LTD., (3) 49-16-95, TLX 341667 **SWEDEN:** AB AKTA, (08) 54-20-20, TLX 13702 **USA:** CONTACT AMPRO COMPUTERS INC., TEL. (415) 962-0230 TELEX 4940302

IBM®, IBM Corp., Z80A®, Zilog, Inc., CP/M®, Digital Research, ZCPR3™ & ZRDOS™, Echelon, Inc., Turbo DOS®, Software 2000, Inc., T/MAKER III™, T/Maker Co

AMPRO COMPUTERS INCORPORATED

67 East Evelyn Ave. • Mountain View, CA 94041 • (415) 962-0230 • TELEX 4940302

Networking—Bus hosts can also use the bus to communicate with each other. In effect, SCSI can be used as a low cost, high speed inter-processor communications channel.

Mass Storage Availability—Not to be overlooked is the myriad of mass storage peripherals available on SCSI, including Winchester disk, tape, optical storage, bubble, RAM-disk, and more.

Single-Chip Bus Interfaces—SCSI bus interface ICs are now available from many sources which implement the entire interface—including bus drivers and receivers—in a single package. In addition, these devices are low cost ($10 to $20).

Host Interface Availability—Nearly every computer bus now has readily available SCSI "host adapters," which serve as gateways to SCSI. Also, both single board computers and closed architecture systems (such as the Macintosh Plus and the new Wang laptop PC) have begun to include a SCSI port as a standard feature.

High Performance—SCSI supports data rates of 1.5 megabytes per second in the normal asynchronous mode, and up to 4 megabytes per second in the optional synchronous mode of data transfer.

SCSI is an Intelligent Bus

One of the major advantages of the SCSI bus is its inclusion of device command protocols in addition to the data transfer signal protocols normally associated with microprocessor buses. Non-identical devices can appear the same to the host computer by virtue of the implementation of one of SCSI's "Common Command Sets." This has a number of important advantages to the host's software, including:

Increased multisourcing of peripheral devices—the common command set, along with the SCSI bus interface and protocols, make it much easier for devices controllers from several manufacturers to function identically from the host system's hardware and software point of view.

Reusability and portability of host driver software—Due to the general similarity of SCSI protocols regardless of device type, once a software driver is writtem for one SCSI device, it is easily modified for another device. Further, programs written for use on one host system can be easily ported to other systems; in fact no porting is required at all if a partitioned software architecture providing an SCSI BIOS extension is utilized once the SCSI BIOS extension is implemented.

Simplification of host driver software development—In addition to the points made above, the host software is greatly simplified by virture of the on-board firmware located on the SCSI device or controller. Typically, only high level commands need be given by the host.

These advantages are not, however without cost. The use of relatively high level command sets to control the peripheral devices requires that the device controller posses sufficient local intelligence to interpret the commands and communicate with the host using SCSI's command protocols. In practice this is accomplished through the use of a single chip microcontroller (e.g. 8041), or a combination of microprocessor (e.g. Z80), RAM, and EPROM.

Additional functions are provided by the SCSI peripheral controller's on board intelligence which generally far outweigh the cost of that intelligence. These include:

Increased peripheral data throughput—SCSI devices

generally provide a degree of on board data buffering in one or both directions which is optimized for the particular type of interface.

Improved real time response— Nearly all SCSI devices have an on board dedicated microprocessor. The device's on board processing power and interrupt support can be tailored according to the requirements of the specific interface.

Reduction of host processing bandwidth requirements—Since the low level maintenance of the interface is under control of the device controller's on board microprocessor, far less load is placed on the host's CPU.

A SCSI Real Time IOP

A link between the SCSI bus and a variety of off the shelf real time interfaces can be created through the use of an intelligent I/O processor which communicates with the host system over the SCSI bus, and implements a low level "internal" bus for interface module connections. Such a device has been called a "SCSI/IOP"® , and is illustrated in Figure 1.

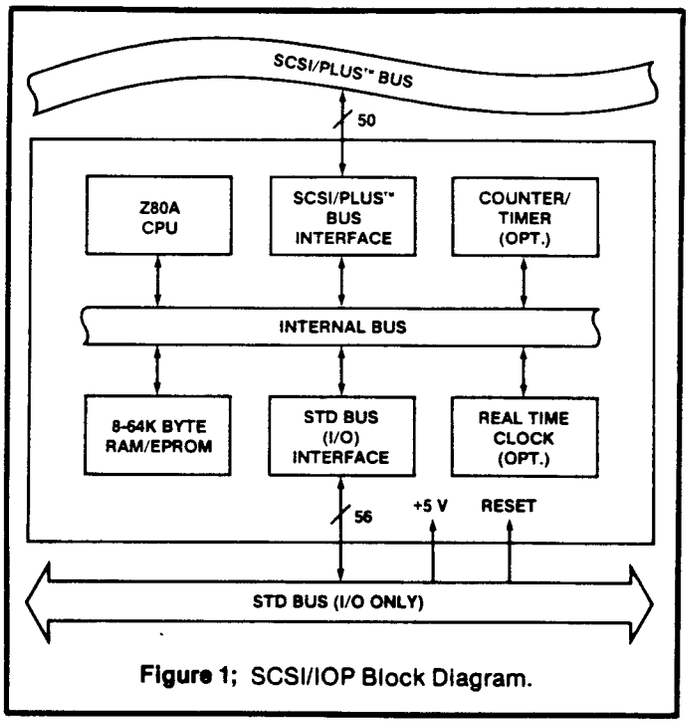


Figure 1; SCSI/IOP Block Diagram.

The "SCSI/IOP" shown in Figure 1 allows any host with a SCSI interface to access the real time interfaces connected to the IOP's "internal" STD Bus. By using STD Bus as an auxilliary I/O bus, a wide variety of low cost data acquisition and control interface can be modularly added to the host system's I/O resources. The 8-bit limitation of STD Bus (as commonly implemented) is not a disadvantage, since the SCSI bus data path is also 8-bits. Furthermore, STD Bus is simple to implement and does not burden the IOP's hardware or software design. (The electronics required for maintenance of the SCSI interface and to interface with the STD Bus requires only a few inexpensive ICs.

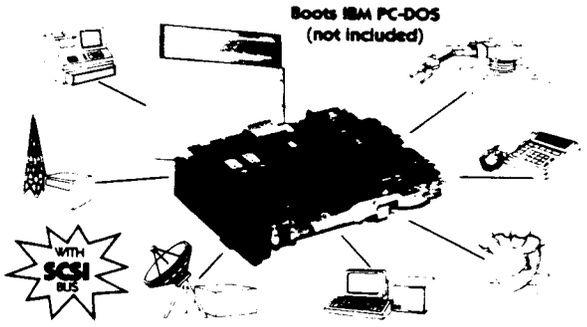
The following ten SCSI commands are currently included in the SCSI/IOP's firmware, though many more can be added:

Request Sense—Returns information regarding the device and the device controller.

SCSI ENGINES

Little Board/186™ $495

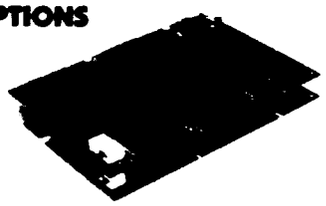
High Performance, Low Cost PC-DOS Engine

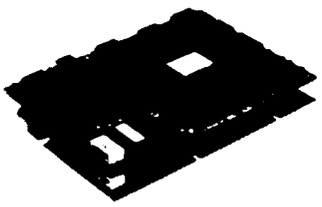


- Three times the COMPUTING POWER of a PC
- Data and File Compatible with IBM PC, runs "MS-DOS generic" programs
- 8 MHz 80186 CPU, DMA, Counter/Timers, 128/512K RAM zero wait states, 16-128K EPROM
- Mini/Micro Floppy Controller (1-4 Drives, Single/Double Density, 1-2 sided, 40/80 track)
- 2 RS232C Serial Ports (50 -38,400 baud), 1 Centronics Printer Port
- Only 5.75 x 7.75 inches, mounts directly to a 5-1/4" disk drive
- Power Requirement: +5VDC at 1.25A, +12VDC at .05A; On board -12V converter
- SCSI/PLUS™ multi-master I/O expansion bus
- Software included:
  - PC-DOS compatible ROM-BIOS boots DOS 2.x and 3.x
  - Hard Disk support

OPTIONS

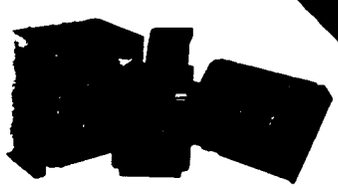
PROJECT BOARD/186™ - adds 25 square inches of wire wrap prototype area with buffered and pre-decoded 80186 bus interface for Little Board/186





EXPANSION/186™ - adds five key options to Little Board/186
- 512K RAM
- 8087 co-processor
- Battery-backed Real Time Clock
- 2 RS232/422 sync/async serial ports
- I/O expansion bus

VIDEO RAM EMULATOR™ -allows use of software that writes to display controller "VIDEO RAM"



SCSI/IOP™ - permits connection of off-the-shelf STD bus industrial I/O interfaces (analog, digital, serial, display, power control, etc.)

DISTRIBUTORS

ARGENTINA: FACTORIAL, S.A., 41-0018 TLX 22408 AUSTRALIA: ASP MICROCOMPUTERS, (613) 500-0688, TLX 36587 BELGIUM: CENTRE ELECTRONIQUE LEMPEREUR, (041) 93-45-41, TLX 49621 BRAZIL: COMPULEADER COMPUTADORES LTDA., (41) 862-1939, TLX 416132 CANADA: TRI-M, (604) 438-9012 DENMARK: DANBIT, (03) 66 90 90, TLX 43558 UK: AMBAR SYSTEMS LTD., 0296 35511, TLX 837487 FINLAND: SYMMETRIC OY, 358-0-585-392, TLX 121394 FRANCE: EGAL PLUS, (1) 4508-1800, TLX 620893 ISRAEL: ALPHA TERMINALS, LTD., (03) 49-16-95, TLX 341667 SWEDEN: AB AKTA, (08) 54-90-90, TLX 13702 USA: CONTACT AMPRO COMPUTERS INC.

IBM®, IBM Corp., 80186®, Intel, Corp.

AMPRO

COMPUTERS, INCORPORATED

67 East Evelyn Ave, • Mountain View, CA 94041 • (415) 962-0230

TELEX 4940302 • FAX (415) 962-1042

Read IOP Memory—Allows the host to directly read the contents of the IOP's onboard memory.

Write IOP Memory—Allows the host to directly write to the IOP's onboard memory.

Branch—Causes the IOP to begin code execution at a specified location in its onboard memory. This permits the use of multiple algorithms or programs downloaded with the HEX Download command.

Read I/O Port—Allows the host to directly read an I/O port on the IOP's I/O device interface.

Write I/O Port—Allows the host to write directly to I/O ports on the IOP's I/O device interface.

Read/Set Clock—Allows the host to read or set the battery backed real time clock.

Read Block—Used by the host to read a block of data from the IOP's RAM or ROM disk option.

Write Block—Used to write a block of data to the IOP's RAM or ROM disk option.

HEX Download—Allows the host to send programs to the IOP for subsequent execution with the Branch command.

Benefits of the SCSI/IOP Architecture to Real Time I/O

Several important benefits result from the SCSI/IOP architecture as a means to standardize small system read time I/O expansion:

Distributed Intelligence— Due to the multi-master feature of SCSI, a single host can have multiple IOP's on its SCSI bus. Up to seven SCSI/IOP's can be controlled by a single host system, providing a high degree of distributed real time data acquisition and control capability. Tasks can be simultaneously run on multiple SCSI/IOP's. In addition, due to the peer-to-peer nature of the SCSI bus, each SCSI/IOP can be programmed to asynchronously communicate with the host system—and with other SCSI/IOP's.

Real Time Capability— A major disadvantage of the most popular disk operating systems (e.g. MS-DOS, UNIX, and CP/M) is their lack of support for real time events. This problem is greatly alleviated by the SCSI/IOP since the IOP contains a coprocessor which can process and even respond to real time stimuli. As an example, the system CPU is generally directly involved in the low level requirements of floppy disk controller support when a sector of data must be read or written from a floppy diskette. This is generally incompatible with the real time requirements of data acquisition and control. Furthermore, many commercial operating systems were developed for desk top applications and have no real time or multi-tasking capability at all.

By offloading the real time chores to the SCSI/IOP, a single board computer such as the AMPRO Little Board/186 can support the requirements of an operating system such as IBM's PC-DOS while the IOP manages the data acquisition and control as a slave coprocessor.

Fault Tolerance— The SCSI/IOP also provides a high degree of protection against system faults in three key ways: (1) The SCSI bus buffers the host from the I/O subsystem's real world interface bus (STD Bus). As a result, failures in the I/O subsystem will not crash the host CPU because the host's internal bus is distinct from the STD Bus. The I/O subsystem can also be powered independently, to further protect the host from I/O subsystem failure.

(2) Multiple SCSI/IOP's can be connected to a single host on the SCSI bus, for redundant I/O subsystems. (3) Multiple hosts can share one or more SCSI/IOP's, allowing redundant hosts as well.

Interchangeability— Finally, one of the most significant advantages of the SCSI/IOP architecture is that it allows real time interface subsystems containing a variety of specific interfaces to be used interchangeably, resulting in such possibilities as multi-sourced data acquisition and control subsystems. In addition the reduction of I/O interface functions to a common set of standardized SCSI commands allows the I/O subsystem to be moved from host to host with little impact on software, especially if the applications are written in a highly transportable language such as C, Pascal, or Forth. As a result, the host processor can be selected according to the needs of the aplication, and can be any type of system having an SCSI bus, including IBM PC, Apple Macintosh Plus, DEC VAX, or single board computers with on-board SCSI such as the AMPRO Little board series.

Conclusions

The current generation of microcomputer system buses bring with them multiple device interconnects, including a variety of new I/O subsystem buses. This article has described an alternative to this approaach, which involves adding SCSI—as a universal I/O bus—to both bus-based and non-bus system architectures. Real time I/O on SCSI is accommodated by means on an I/O processor such as the SCSI/IOP. This approach has many advantages, including:

- Addition of real time capability to computers and operating systems which are not normally strong in that area.
- Distributed processing.
- Fault tolerance in both hardware and software.
- Reduction of software development efforts.
- Improved portability of software.
- Increased availability of multi-sourced I/O interfaces. ■

An Open Letter to STD-Bus Manufacturers

Getting an Industrial Control Job Done

By Jerry Nelson, Ph.D., Phillips University

Our Common Ground

We are not trying to finalize the Manufacturing Automation Protocol for a steel mill or the Technical Office Protocol for a building full of Boeing engineers. We are the other people who make machines and research labs go. Our common ground is a pile of STD cards and Opto 22® modules wired to the real world, with a CPU card telling the rest of them what to do. And we want to write programs for this CPU on a powerful, comfortable computer full of software tools.

Unity of Development and Target Systems

We have all discovered something even nicer than a program development system with in-circuit emulation and a hardware logic analyzer. It is a computer matched to the target. If you write on a PC, and the target pile of STD cards also runs like a PC, you can take your program from one and try it out on the other.

Great Things Are Happening for STD at the 16-bit level

Rather than having just a 16-bit CPU card on the STD bus and an umbilical cord linked to a powerful, comfortable computer somewhere else, it is now possible to build a PC clone on the STD bus, with cards from ProLog, and from ZiaTech. ZiaTech also offers EGA graphics on an STD card, based on the Paradise Auto-Switch chip set. The PC development package is completed with the VRTX® multi-tasking executive licensed from Hunter & Ready. Half of the two problems with this package—high cost and complexity—have been solved by ZiaTech by pre-configuring the executive to their hardware and to the MS-DOS® operating system which runs under it. The configured VRTX, in which MS-DOS runs as one of the tasks, is called Multi-DOS.

Sit down anywhere to program, even next to the pile of STD cards, and just slip a disk into the target to bring it to life, test the application.

One thing that is NOT happening nicely at the 16-bit level is the expansion of the PC itself for industrial process control. The A/D D/A data acquisition add-in cards are legion (Data Translation, Asyst, LabTech Notebook and all that), but real-world control cards are limited and there is limited space to put them inside a PC. The industry leader for such cards is Metrabyte, and Metrabyte tried to solve the expansion problem. Their solution was a proprietary bus, and we hope the customers they trap with it will be happy.

Industrialized, SBC-style PC-ATs in bigger, ruggedized boxes (Faraday, Sigma Information Systems) is a more rational approach because it is based on an open bus. But the PC bus is the wrong bus. With STD, we already enjoy all the functions we need, on the right-size real estate; there is little money to be made re-inventing this wheel on PC boards.

A PC with STD-bus Expansion —Who has linked the PC to the STD bus? Not as a terminal to a target, but as one system? A SCSI port in the PC can drive an external STD expansion bus. At the STD end, the ribbon cable stops at an Ampro SCSI/IOP intelligent I/O processor. Back at the PC end, all one needs is the realization that SCSI is not just for Winchesters. SCSI is universal, SCSI supports Very Local Area Networks (distributed computer systems). Call Rick Lehrbaum, Vice President of Engineering at AMPRO, for advice on how to do it. The practice you get will help you apply SCSIs to the office market, where its 1 Mbyte/s transfer rate meshes nicely with the 1.05 Mbyte size of a 300 dpi laser printer page image.

A PC/STD link using LAN rather than SCSI technology was offered 1/85 by Beal Communicaions, based on a token bus protocol similar to Arcnet. How are they doing in the marketplace? What are the application areas?

"Simplicity and completeness are everything for people interested in solutions to problems, ..."

Great Things Are Waiting to Happen for STD at the 8-bit Level

We need: (1) A powerful, comfortable computer for program development, (2) Tools, especially a multi-tasking executive, and (3) Access to the STD bus.

Software— The tools are here: 8-bit Modula-2 from Borland International, and the Quick Task real-time multi-tasking executive from Echelon, Inc. I call your attention to Quick Task, and invite comparison with Hunter & Ready VRTX, Kadak, U.S. Software, and IPI MTOS/MP.

Hardware— The 8-bit renaissance has brought power with a VLSI equivalent to the 80186, namely the Hitachi HD64180. Computers using it include the SemiDisk's DT-42 and the MicroMint SB-180FX. We are at the threshold of 1 Mbyte RAM directly addressed from a 12.288 MHz CPU chip. To slow down the powerful development computer when addressing industrial STD peripheral cards, I call your attention to the TA84HC04 I/O cycle extender chip from Texas Arrays, Inc.

Operating System— The powerful development computer is comfortable because of its new operating system, to which a UNIX-like assortment of tools automatically configure themselves. This is the total CP/M® replacement called ZDOS®, from Echelon.

A weakness here, as with the PC-AT, is multi-tasking—in the comfortable development environment, not in the target application Until multi-tasking ZDOS comes from Richard Conn and Echelon. there is rapid context switching and restoration using Backgrounder II by Bridger Mitchell, from NAOG, the North American One-Eighty User's Group.

What are We Waiting For?— We are waiting for someone to put a full-up HD64180/ZDOS system on the STD bus, or an STD expansion bus link to an existing HD64180/ZDOS computer.

On the STD bus we have Hitachi HD64180-based CPU cards from several suppliers but no turn-key systems (c'mon guys, I want to build my application, not the development system too).

Off the STD Bus we have from AMPRO a SCSI link to the STD, but the machine they link well is an 80186; they link the 8-bit Z80 poorly (you have to write the software; the card needs a piggyback add-on in the CPU socket to retrofit the SCSI port); and the HD64180 they link not at all. Their product line has PC-compatible 80186s, Z-80s, and a hole in the middle where the HD64180 should be.

Editor's note: AMPRO now makes the Little Board/Plus which incorporates the SCSI on the main board and eliminates the need for a daughter board, and support software is available for their SCSI/IOP.

We have an HD64180 SBC with SCSI port from Southern Pacific Limited in Japan, but they use the dead end CP/M+ operating system, not ZDOS and offer no STD bus expansion.

We have all the Right Stuff from Micromint, but they haven't gotten around to developing and marketing a system with an STD cage attached to their SCSI port.

Conclusion

Simplicity and completeness are everything for people interested in solutions to problems, not software "features," hardware "tweaks," or ideological arguments.

Connecting the STD rack via SCSI, either to a PC or to the 8-bit renaissance, gives the STD bus the completeness it needs for market stability. The technology is available from Ampro. This connection gives the growing number of cost-effective, bus-less Single Board Computers (SBCs) now entering the market a universal route to expansion. Expansion could be to STD cards in industrial/laboratory control, laser printers, Winchesters. Let's make it official for expanding computers: **'If it isn't a PC, give it an STD.'**

Putting the entire PC or entire super 8-bit machine on the STD bus is useful and even necessary for some applications. But it will always be less cost-effective than links to the powerful, comfortable machines the engineer otherwise chooses and uses. In other words, machines that sell to the offices and everyone else will always be cheap and can have industrial I/O aded; special machines configured on an industrial bus will always cost more.

Please send me your wisdom; tell me about products I have forgotten. I can't wait to be told this market snap-shot is out of date. If I hear about enough new items, I will revise this open letter. Initial circulation was confined to the list of suppliers mentioned.

Jeremih I. Nelson, Ph.D.
Working Group in Biophysics
Philipps University; Renthof 7
D-3550 Marburg
WEST GERMANY
Telephone 06421/284161

Suppliers List

Rick Lehrbaum
Ampro Computers, Inc.
67 East Evelyn Ave.
Mountain View, Ca 94039
SCSI/IOP I/O processor, links STD bus to powerful, comfortable development computer over a SCSI port. Their line includes SBCs and turnkey systems with ZDOS but not the Hitachi HD64180 CPU.

Beal Communications
11020 Audilia Road
Suite C101
Dallas, TX 75243
NETPC/STD card uses token bus protocol similar to Arcnet to link PC to STD cage for industrial control applications.

John Heiner or Mike Nicewonger
CuBit
190 SO. Whisman Rd.
Mountain View, CA 94041 They make a nice all-CMOS STD HD64180 CPU card, but have no intention of offering a complete system. Their customers want to connect PCs to the STD bus even if CPUs are different and cross-software is needed.

DYAD Technology Corp.
4040 Sorrento Valley Blvd.
San Diego, CA 92121
VRTX on $1,700 add-in card for the PC guarantees more complete, ready-to-go configuration than the software-only DSP from Hunter & Ready.

Frank Gaude
Echelon, Inc.
885 North San Antonio Road
Los Altos, CA 94022
The 8-bit renaissance: ZDOS operating system and utilities; Quick-Task multi-tasking executive, distribution of Borland Int'l MODULA-2.

Hunter & Ready, Inc.
445 Sherman Avenue
Palo Alto, CA 94306
VRTX versatile real-time executive; DSP, "Device support package" for IBM PC available.

Industrial Programming, Inc.
100 Jericho Quadrangle
Jericho, NY 11753
MTOS-80MP multi-processor, multi-tasking executive for 8080 & up; this family of compatible execs is especially popular in the 68000 world.

Kadak Products, Ltd.
206-1847 West Broadway Ave.
Vancouver, BC V6J 1Y5 CANADA
AMX Multitasking Executive for Z80, etc.

Laboratory Technologies, Inc.
255 Ballardvale St.
Wilmington, MA 01887
LABTECH NOTEBOOK for data acquisition; available through Analog Devices and Burr-Brown. Data acquisition systems in the PC can not grow to industrial and lab controi systems, but create the market for them.

M.A.N. Systems
323 N. 3rd.
Medford, OK 73759

As long as Ampro won't make an HD64180, one must turn instead to this an HD64180+RAM add-on for the Ampro Z80 card.

MacMillan Software, Inc.
630 3rd Ave.
New York, NY 10022
Asyst, and Asystant Plus Software for A/D D/A cards. The engineers and scientists introduced to data acquisition this way now want expansion to industrial and lab control.

MetraByte Corp.
440 Myles Standish Blvd.
Taunton, MA 02780
Beyond A/D D/A cards for data acquisition; Opto22 and the proprietary MetraBus driven from the IBM-PC.

Ray Alderman Matrix
1639 Green Street
Raleigh, NC 27603
If they made a 64180, it would be nice.

Craig Henderson
MicroAide
685 Arrow Grand Circle
Covina, CA 91722
Working on release of their STD board with Hitachi 64180 CPU, WD 33C93 FDC and ZDOS operating system. Who will be the value-added OEM to offer a turnkey system based on this card?

MicroMint, Inc.
#4 Park Street
Vernon, CT 06066
The SB-180 of BYTE/Steve Ciarcia fame. Piggyback SCSI links hard disk, but not STD bus.

Mr. Frank Wrobel
Microtrix, Australia
24 Bridge Street
Eltham, VIC 3095
AUSTRALIA
HD64180 & ZDOS on STD bus; startup company without turnkey system as of 11/86. Please tell me the current status on the Microtrix product.

Bruce Morgan
North American One-Eighty Group
PO Box 2781
Warminster, PA 18974
Source for Bridger Mitchell's Backgrounder II (rapid context switching in the new Hitachi 64180/ZDOS world).

Mr. Byron Brooks, VP Marketing
Mr John Carobine, Mgr., Customer Support
Oneac Corp.
27944 N. Bradley Road
Libertyville, IL 60048
An elegant new 8-bit machine with ZDOS, a company without Hitachi. Engineers are among those who would appreciate this totally battery backed, big RAM, diskless machine—the only way to loose data on this computer is to throw it out the window. But engineers need both a 64180 host and a SCSI link to target STD card cage.

ProLog, Inc.
2411 Garden Road
Monterey, CA 93940
PC on the STD bus, but ZiaTech has the glory. ProLog should re-think their strategy: 8AND 16, not 8OR 16. Market turnkey systems on the bus and off the bus.

Scientific Solutions, Inc.
6225 Cochran Road
Cleveland, OH 44139
Lab Master, Lab Tender A/D D/A acquisition cards for PCs.

SemiDisk Systems, Inc.
PO Box GG
Beaverton, OR 97075
Their DT42 is a lovely HD64180 SBC board with ZDOS operating system, integrated to the famous SemiDisk 8Mbyte battery backed RAM card for a disk-less system. Who sells this as a turnkey system? A link to the STD bus is needed for industrial and lab control use.

Sigma Information Systems
3401 E. LaPalma Ave.
Anaheim, CA 92806
Industrialized version of PC-AT; SBC plugs into cage.

Softaid, Inc.
8930 Route 108
Columbia, MD 21045
64180 MT-Basic multi-tasking BASIC compiler (same company as Z80 I.C.E. Box tester).

Hiroshi Katayama, President & Director
Southern Pacific Limited
Sanwa Bldg.
2-16-20 Minamisaiwai
Nishi, Yokohama
JAPAN 220
MSC-LAT1 & -LAT2 computers, HD64180 with CP/M +, not ZDOS.

Hitachi America, Ltd.
2210 O-Toole Ave.
San Jose, CA 95131
HD64180 chip; HD63484 graphics controller.

Texas Arrays, Inc.
1301 Dentron Dr.
Carrollton, TX 75006
TA84HC04 integrates slow Z80 I/O support chips with fast 8-bit renaissance CPUs.

U.S. Software
Microcontroller Dvsn.
5470 N.W. Innisbrook Place
Portland, OR 97229
USX Executive for 8086, 8051, 8096; Multi-Tasking Kernal for everything (Z80, 6502, 68000 family).

Gary Harris
VersaLogic Corp.
87070 Dukhobar Road
Eugene, OR 97402
An elegant concept: a Z80 Smart Card with NOS (no disk non-volatile operating system) and C4 BASIC. "Write and test programs in electronically erasable PROM or 'self-powered' static RAM chips. No assembling, compiling or even PROM burning. Programs are there to stay, with or without power. Copy, remove and transport the chips like regular PROMS."
Perhaps we could update this elegant concept with a 64180 CPU and 64180 multi-tasking BASIC from Softaid, Inc. Instead of a NOS, use Oneac's and SemiDisk's concept of disk emulation in cheap DRAM. Now all out tools run on a Very Smart Card, and we can plug in rotating disks at any time.

Vrooman Chan Communications
P.O. Box 5822
Station A
Toronto, ONT M5W 1P2
CANADA
A 6.5 × 4.0" Hitachi SBC. Why not rework it as a 6.5 × 4.5Pvf0 STD card with attached 3.5" microfloppy as a complete ZDOS industrial system.

Bob Burkle or Jerry Winfield
WIN Systems
PO Box 121361
Arlington, TX 76012
100% CMOS HD64180, but no FDC, no turnkey systems; do we really want to run cross-software from MicroTech Research on DEC minis to generate 64180 code?

ZiaTech, Inc.
3433 Robert Court
San Luis Obispo, CA 93401
PC with EGA on STD bus; "Multi-DOS": VRTX configured to this PC and to MS-DOS. The most sophisticated and complete example of a 16 bit development system on the STD bus.

Editor's Note: I slightly condensed th list to fit the space available, and ar errors or typos are my responsibility.

(Continued from page 3)

learned how to achieve the maximum from what we already have. There are instances where nothing but the newest will do, such as the 68030 or 80387 with a math coprocessor for heavy number crunching, but we should not try to fool ourselves by saying that existing equipment can not continue to do what it already does well. We have to be frank and just plain admit that a lot of the fun is learning about new systems—that's why I have one of Hawthorne's TinyGiant 68000 SBCs, because I want to learn 68K assembly language and how to interface the 68K peripheral chips for control applications.

Our knowledge of several systems will enable us to recognize the strengths and weaknesses of the different systems so that we can choose the best solution for a particular application, and even devise new systems combining the best features of the ones we have used.

It's Not Software or Hardware It's the Solutions to Problems

A number of people picked up on the above statement in the last issue, and we are going to emphasise using computers to solve problems. This will involve the approaches of selecting the best system for the application when equipment can be purchased for the specific use, and making existing equipment work when time and/or money limitations prevent you from buying new equipment.

I don't plan on ever working with just one system! I'll keep the old while adding the new, and I'll be very selective about using the most suitable system for the application because I'll have a lot from which to choose.

I will be adding an IBM PC XT or AT clone with a 20 Meg hard drive because I need the low-cost high-quality accounting, spreadsheet, cross assemblers and cross compilers, desk top publishing, and engineering software which are available for this system. I just can't continue without the 16-bit engineering software available from BV Engineering (2200 Business Way, Suite 207, Riverside, CA 92501, (714) 781-0252), and if you're interested in anything more than wordprocessing and spreadsheets, you should get their free catalog. I also need to add CAD (Computer Aided Drafting) and CAE (Computer Aided Engineering) to prepare schematics, illustrations, and printed circuit board layouts.

The fact that I'm finally getting a PC does not mean that I will stop using my Ampro Z-80 Little Boards with Echelon's ZCPR3. I still feel that the Z-80 systems are the best choice for many measurement and control uses, and when I need higher performance I'll use Hawthorne's 68000 TinyGiant. For specific uses such as ground water table level logging I'll even use 8-bit microcontrollers instead of a full microcomputer, and the data can be collected and analyzed by almost anything. I'm not enthused about learning all about low-level PC DOS programming or assembly language programming for the 8088's segmented architecture—I'll primarily stick with the above mentioned software and a few simple utilities in Pascal or C, and do the rest of my work on the Z-80 or 68000. I realize how important the PC is for those working in that market; and we will publish serious PC programming articles, even though I do not choose to work in that environment myself.

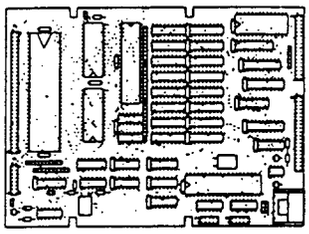
Computer Controlled Machining

I'm finally getting started on my long delayed project of adapting my 12" Atlas

68000 SINGLE BOARD COMPUTER

$395.00

32 bit Features / 8 bit Price



-Hardware features:
* 8MHZ 68000 CPU
* 1770 Floppy Controller
* 2 Serial Ports (68681)
* General Purpose Timer
* Centronics Printer Port
* 128K RAM (expandable to 512K on board.)
* Expansion Bus
* 5.75 x 8.0 Inches Mounts to Side of Drive
* +5v 2A, +12 for RS-232
* Power Connector same as disk drive

Add a terminal, disk drive and power, and you will have a powerful 68000 system.

-Software Included:
* K-OS ONE, the 68000 Operating System (source code included)
* Command Processor (w/source)
* Data and File Compatible with MS-DOS
* A 68000 Assembler
* An HTPL Compiler
* A Line Editor

ASSEMBLED AND TESTED ONLY **$395.00**

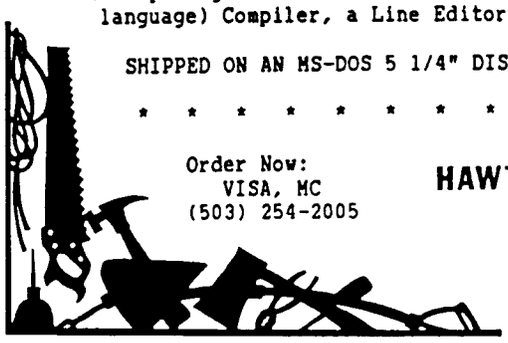
* * * * * * * * * * * * * * * * *

K-OS ONE, 68000 OPERATING SYSTEM

For your existing 68000 hardware, you can get the K-OS ONE Operating System package for only $50.00. K-OS ONE is a powerful, pliable, single user operating system with source code provided for operating system and command processor. It allows you to read and write MS-DOS format diskettes with your 68000 system. The package also contains an Assembler, an HTPL (high level language) Compiler, a Line Editor and manual.

SHIPPED ON AN MS-DOS 5 1/4" DISK. **$50.00**

* * * * * * * * * * * * * * * * *



Order Now:
VISA, MC
(503) 254-2005

HAWTHORNE TECHNOLOGY

8836 S.E. Stark
Portland, Or 97216

lathe for computer controlled machining, and I would like to hear from anyone with helpful suggestions (or even those with similar problems).

I want to drive the leadscrew, the crossslide, and the compound slide under computer control, and the first step will be to determine where I can use stepper motors and where I will need to use DC motors. I plan on incorporating shaft encoders and closed loop feedback control systems (see BV Engineering's article in TCJ#27) to ensure that the requested operation really occurs. One of the major problems is the mechanical installation of the motors and sensors because I want to retain the hand control feed wheels and haven't figured out how to attach the motors. This is a mechanical problem instead of a computer problem—but that's what you run into when working with real world devices.

Once I have it running I'll develop programs with logic seeking for taper and contour turning, and threading, and then add special routines for cross slide mounted drilling, milling, and grinding attachments. Future enhancements will be a computer controlled quick change tool post, a stepper motor driven dividing head for the cross slide, and a stepper motor dividing head control for the spindle to be used in conjunction with the cross slide mounted drilling and milling attachments.

My programming language choice for this project will be C for the user interface with the functions written in assembler or C depending on the speed required. I'll use BDS C (version 1.6) because it has a good library manager, it is easy to incorporate assembly language functions, and it will produce ROMable code. I anticipate that this will require several microprocessors and/or microcontrollers for real time control and that the STD Bus would be ideal, but I'll probably start with several Ampro 8-bit Little Boards plus wirewrapped boards and go with the flow as things develop—perhaps I'll end up using one of my S-100 systems.

While working on this project I have noted the scarcity of information on low cost motors, drivers, encoders, and the special chips suitable for the low volume user. We will be establishing resource information on the availability of hardware and software suitable for the experimenter or developer, and any information that you can supply will be much appreciated. The best sources that I have for motors and hardware at this time are Silicon Valley Surplus, United Products, and Jerryco. I would be especially interested in public domain stepper or DC motor driver programs including ramp-up and ramp-down features, and feed back control programs. This information and software will be placed on the BBS if there is sufficient response.

C Resources

One of the results of my working with many different systems is that I can not afford to buy a full set of commercial software for every system, so I depend on using public domain utilities where ever possible—and I'm only interested in programs with the source code so that I can modify, incorporate in programs, link with programs, add to special libraries, etc.

The best source of programs which fit my needs is CUG (The C User's Group, Box 97, McPherson, KS 67460, phone (316) 241-1065), because they offer useful utilities for both 8-bit and 16-bit systems with the source supplied on most disks and they also publish a quarterly newsletter which serves as an international resource for C programmers. To make their collection even more valuable, they have recently produced a Library Directory of disks up to #199 which makes selection of the functions you need easy. This directory is $10 and is well worth the price.

In order to support CUG, we will be listing the contents of their new releases, and reviewing selected programs. We would like to publish additional C programming manuscripts or reports on the user's disks, so contact us if you're interested.

Another valuable resource is the DESMET GAZETTE (Pacific Data Works, 1341 Ocean Ave., #257, Santa Monica, CA 90401, phone (213) 390-4854) published by Andrew Binstock. Even though the main subject is Desmet C, there is a lot of information useful to all C programmers so that it is well worth the $18 per year. ■

TURBO PROGRAMMERS–

TDebug Plus™...

...CUTS DEBUGGING FRUSTRATION.

TDebugPLUS is a **new** interactive symbolic debugger that integrates with Turbo Pascal to let you

- **Examine and change variables** at runtime using symbolic names – including records, pointers, arrays, and local variables.
- **Trace and set breakpoints** using procedure names or source statements.
- **View source code** while debugging.
- **Use Turbo Pascal editor and DOS DEBUG commands**

TDebugPLUS also includes a special MAP file generation mode fully compatible with external debuggers such as Periscope, Atron, Symdeb, and others – even on programs written with Turbo EXTENDER.

An expanded, supported version of the acclaimed public domain program TDEBUG, the TDebugPLUS package includes one DSDD disk, complete source code, a reference card, and an 80-page printed manual. 256K of memory required. Simplify debugging! $60 COMPLETE.

TURBO EXTENDER™

Turbo EXTENDER provides you the following powerful tools to break the 64K barrier:

- **Large Code Model** allows programs to use all 640K without overlays or chaining, while allowing you to convert existing programs with minimal effort, makes EXE files.
- **Make Facility** offers separate compilation eliminating the need for you to recompile unchanged modules.
- **Large Data Arrays** automatically manages data arrays up to 30 megabytes as well as any arrays in expanded memory (EMS).
- **Additional Turbo EXTENDER tools** include Overlay Analyst, Disk Cache, Pascal Encryptor, Shell File Generator, and File Browser.

The Turbo EXTENDER package includes two DSDD disks, complete source code, and a 150-page printed manual. Order now! $85 COMPLETE.

TURBOPOWER UTILITIES™

"If you own Turbo Pascal, you should own TurboPower Programmers Utilities, that's all there is to it." Bruce Webster, **BYTE Magazine**

TurboPower Utilities offers nine powerful programs: Program Structure Analyzer, Execution Timer, Execution Profiler, Pretty Printer, Command Repeater, Pattern Replacer, Difference Finder, File Finder, and Super Directory.

The TurboPower Utilities package includes three DSDD disks, reference card, and manual. $95 with source code; $55 executable only.

ORDER DIRECT TODAY!

- **MC/VISA Call Toll Free** 7 days a week. 800-538-8157 x830 (US) 800-672-3470 x830 (CA)
- **Limited Time Offer!** Buy two or more TurboPower products and save 15%!
- **Satisfaction Guaranteed** or your money back within 30 days.

For Brochures, Dealer or other information, PO, COD – call or write:

TURBO Power™ 3109 Scotts Valley Dr., #122 Scotts Valley, CA 95066 (408) 438-8608 M-F 9AM-5PM PST

The above TurboPower products require Turbo Pascal 3.0 (standard, 8087, or BCD) and PC-DOS 2.X or 3.X, and run on the IBM PC/XT/AT and compatibles.

Back Issues Available:

Issue Number 1:
- RS-232 Interface Part One
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an "Epram"

Issue Number 2:
- File Transfer Programs for CP/M
- RS-232 Interface Part Two
- Build Hardware Print Spooler: Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:
- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for the Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler: Part 2

Issue Number 4:
- Optronics, Part 1: Detecting, Generating, and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler: Part 3
- Beginner's Column: Power Supply Design

Issue Number 8:
- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board: Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 14:
- Hardware Tricks
- Controlling the Hayes Micromodem II from Assembly Language, Part 1
- S-100 8 to 16 Bit RAM Conversion
- Time-Frequency Domain Analysis
- BASE: Part Two
- Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part 2

Issue Number 15:
- Interfacing the 6522 to the Apple II
- Interfacing Tips & Troubles: Building a Poor-Man's Logic Analyzer
- Controlling the Hayes Micromodem II From Assembly Language, Part 2
- The State of the Industry
- Lowering Power Consumption in 8" Floppy Disk Drives
- BASE: Part Three

Issue Number 16:
- Debugging 8087 Code
- Using the Apple Game Port
- BASE: Part Four
- Using the S-100 Bus and the 68008 CPU
- Interfacing Tips & Troubles: Build a "Jellybean" Logic-to-RS232 Converter

Issue Number 17:
- Poor Man's Distributed Processing
- BASE: Part Five
- FAX-64: Facsimile Pictures on a Micro
- The Computer Corner

Interfacing Tips & Troubles: Memory Mapped I/O on the ZX81

Issue Number 18:
- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1
- The Computer Corner

Issue Number 19:
- Using The Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part Seven
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking and Windows with CP/M: A Review of MTBASIC
- The Computer Corner

Issue Number 20:
- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering and Other Strange Tales
- Build a S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K
- The Computer Corner

Issue Number 21:
- Extending Turbo Pascal: Customize with Procedures and Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition and Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC
- The Computer Corner

Issue Number 22:
- NEW-DOS: Write Your Own Operating System
- Variability in the BDS C Standard Library
- The SCSI Interface: Introductory Column
- Using Turbo Pascal ISAM Files
- The AMPRO Little Board Column
- The Computer Corner

Issue Number 23:
- C Column: Flow Control & Program Structure
- The Z Column: Getting Started with Directories & User Areas
- The SCSI Interface: Introduction to SCSI
- NEW-DOS: The Console Command Processor
- Editing The CP/M Operating System
- INDEXER: Turbo Pascal Program to Create Index
- The AMPRO Little Board Column
- The Computer Corner

Issue Number 24:
- Selecting and Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assembly Code for CP/M
- The C Column: Software Text Filters
- AMPRO 186 Column: Installing MS-DOS Software
- The Z Column
- NEW-DOS: The CCP Internal Commands
- ZTIME-1: A Realtime Clock for the AMPRO Z-80 Little Board
- The Computer Corner

Issue Number 25:
- Repairing & Modifying Printed Circuits
- Z-Com vs Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro Little Board
- Building a SCSI Adapter
- New-DOS: CCP Internal Commands
- Ampro '186: Networking with SuperDUO
- ZSIG Column
- The Computer Corner

Issue Number 26:
- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside AMPRO Computers
- NEW-DOS: The CCP Commands Continued
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS
- The Computer Corner

Issue Number 27:
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis and Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program for Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying The CP/M Disk Parameter Block for Foreign Disk Formats
- The Computer Corner

Wanted

- **Cross Assemblers & Cross Compilers** — We need Public Domain programs with source code, and tutorials on how to write a simple cross assembler for a micro-controller chip.

- **Terminal Control Codes** — TCJ will publish a reference listing of terminal control codes, and also place the data on our BBS. Send the data on your terminal to share with others, and let us know what terminal data you need.

- **C and Pascal Peripheral Modules** — TCJ will publish Pascal Include files and C header files for implementing peripherals such as printers, terminals, etc., and also place this data on our BBS. Send your files to share.

TCJ ORDER FORM

Subscriptions		U.S.	Canada	Surface Foreign	Total
6 issues per year ☐ New ☐ Renewal	1 year	$16.00	$22.00	$24.00	
	2 years	$28.00	$42.00		
Back Issues		$3.50 ea.	$3.50 ea.	$4.75 ea.	
Six or more #'s		$3.00 ea.	$3.00 ea	$4.25 ea.	
				Total Enclosed	

All funds must be in U.S. dollars on a U.S. bank.

☐ Check enclosed ☐ VISA ☐ MasterCard Card # _____

Expiration date_____ Signature _____

Name _____

Address _____

City_____ State_____ ZIP _____

The Computer Journal

190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406) 257-9119

News

I/O Add-On Board for the SB180

Electronic Technical Services (ETS), in cooperation with North American One-Eighty Group (NAOG) and XSystem Software, has announced a hardware and software enhancement for the Micromint SB180. The ETS-180-IO+ is a full-function input/output board, combining all the "must-have" features necessary to make the SB180 a versatile computing engine for efficient software development, personal computing, or real time applications.

The all CMOS ETS-180-IO+ provides—Two additional High speed (115.2 kbps) serial ports with CPU independent baud rates and full hardware handshaking—Twenty-four bits of user configurable parallel I/O—Special version of the XBIOS banked operating system with setup and format utilities, full Z-System support alternate bank disk cache and optimized (2.5 to 5K larger) TPA memory—Complete SASI/SCSI interface, includes XBIOS support for all popular SCSI compatible hard disk controllers and drives—Battery-backed real time clock, with DateStamper time and date stamping of disk files fully integrated into XBIOS—Full buffering of the expansion bus for other add-on boards—Comprehensive interrupt support—All CMOS chip complement for low power operation.

The ETS-180-IO+ is available to individual users through NAOG for $349.95 complete with software, and is available for a limited time for an introductory price of $299.95 including a one year membership in NOAG. Current NAOG members can purchase the ETS-180-IO+ for $284.95. For more information, call Bruce Morgan at (215) 443-9031 or write to: NAOG/ETS, PO Box #2781, Warminster, PA 18974

SB180 is a trademark of The Micromint, Inc.

ETS-180-IO+ is a trademark of Electronic Technical Services

NAOG is a trademark of Bruce Morgan

XBIOS is a trademark of XSystem Software

Z-System is a trademark of Echelon, Inc.

The DateStamper is a trademark of Plu*Perfect Systems, Inc.

QPARSER+ Language Translator

QCAD Systems has announced their QPARSER+ program generator for computer language and translator development.

A translator writing system is a program generator which can be used in the development of any language intensive programming project such as a compiler, assembler, a language to language translator (Pascal to C for example), a file-format converter, a complex user interface, etc,. Other examples are the development of hardware control languages, or interpreters.

William A. Barrett, a member of the QCAD team, is a coauthor of the book *"Compiler Construction"* published by Science Research Associates (ISBN 0-574-21765-7) which is a very useful reference for use with QPARSER+

QPARSER+ is available for the IBM PC, the VAX, or the Macintosh, from QCAD Systems, Inc., 3333A Octavius Drive, Santa Clara, CA 95054, phone (408) 727-6671.

ACNAP3 Circuit Analysis Program

ACNAP3 is an easy to use, general purpose electronic circuit analysis program which analyzes circuits consisting of resistors, capacitors, inductors, transistors, operational amplifiers, FETs, transformers, etc. Circuits up to 200 components and 50 nodes can be analyzed in a single pass. Larger circuits can be chained. ACNAP3 features unlimited length user definable macro operations, double precision accuracy, unattended AUTO execute and BATCH mode operations, subcircuit capability, iterate component value capability, execution of external programs from within ACNAP3, and component libraries. ACNAP3 automatically computes the magnitude, phase and delay at any node in the circuit and includes Global Worst Case, Monte Carlo, Input and Output Impedance calculations, Noise Equivalent Bandwidth and Sensitivity analyses. Free format input, interactive menus, and input error trapping make ACNAP3 easy to learn and use. Low cost graphics modules can be added to upgrade ACNAP3 to drive high resolution graphics screens, graphics printers, and pen plotters.

ACNAP3 is available in PC/MSDOS and Apple Macintosh versions for $125 from BV Engineering, 2200 Business Way, Suite 207, Riverside, CA 92501 phone (714) 781-0252.

FREE CATALOG AND SOFTWARE APPLICATIONS GUIDE

CP/M MSDOS

AFFORDABLE ENGINEERING SOFTWARE

TRSDOS PCDOS

ANALYSIS

XFER $72.95
- Transfer Function Analysis
- Synthesize Circuits/Functions
- Monte Carlo Sensitivities

LOCIPRO $72.95
- Root Locus Analysis
- Up to 26th Order
- Transients (with SPP)

SPP $72.95
- Signal Processing
- FFT, InPlace
- Transient Analysis

STAP $72.95
- 2D Thermal Analysis
- 1600 Nodes

CIRCUIT DESIGN

ACTFIL $72.95
- Active Filter Design
- Low/High Band pass/Bandreject
- Chebychev/Butterworth
- Magnitude Phase Delay

ACNAP $72.95
- AC Circuit analysis
- 30 Nodes 200 Components
- Monte Carlo Sensitivities
- Transients (with SPP)

DCNAP $72.95
- DC Circuit Analysis
- 30 Nodes 200 Components
- Monte Carlo Sensitivities
- ACNAP Compatible Files

GRAPHICS

PDP $72.95
- Pen Plotter Driver
- Linear Logarithmic

PCPLOT $72.95
- High Resolution Graphics
- Linear Logarithmic
- Bit Map Printer Dump

PLOTPRO $72.95
- Generic Graph Printing
- Any Computer Printer
- Long Plots

TEKCALC $72.95
- Scientific Calculator
- Statistics Graphics

MATHEMATICS

TEKCALC $72.95
- Scientific Calculator
- Statistics/Graphics

MATRIX MAGIC $72.95
- Matrices up to 20 x 20
- Operations and Tests

COMCALC $72.95
- Communications Systems design and analysis spreadsheet and Calculator

REPORT WRITING

Right Writer $97.95
- Intelligent Proofreader
- Checks Structure Syntax Punctuation Spelling and more

BV Engineering
Professional Software

2200 Business Way, Suite 207
Riverside, CA 92501 USA

(714) 781-0252

Programming Style

User Interfacing and Interaction

by Art Carlson

One of the first questions a user has when starting with a new program is "How do I enter the parameters which the program needs in order to work?" And some of the questions a software designer needs to address are "How many parameters are involved, are some of them defaults which the user should be able to change, what is the skill level of the potential user, and how often will the program be used?"

I used the term "software designer" because the structure of the software should be designed by someone who considers how the user, the program, and the hardware interact with each other. While the the software is often designed and programmed by the same person, or a small group of persons, this is not the best arrangement because programmers (myself included) are usually so concerned with the technical programming details that they fail to fully consider the user. After all, it's obvious to them (the programmers) how the parameters should be entered, so why shouldn't it be just as obvious to the the novice first time user?

I won't use the term 'user friendly' because it has been overused—even for some programs which are user frustraters. I'm sure that you can supply your own list of programs which run from the extremes of either offering no hint of how to enter the parameters to the opposite extreme of holding you in such a tight over-friendly hug that you can't break out. User/program interaction is a design function at least as important as the problem solving algorithms, but I have seen very little practical material on how to design programs which work with the user, giving them the help they need but not more than they want. Yes, it is possible to provide too much help—I don't like programs which lock me in a friendly shell with level after level of menus when I already know where I want to go.

Menus or Command Line Parameters

Most useful programs require some information about which files to work with or which operations to perform (called parameter passing), and the two most frequently used methods are command line parameters and menus.

An example of the use of command line parameters is the PC DOS copy command where you would enter**COPY A:THISFILE B:THATFILE** which copies the file named THISFILE from drive A: to a file named THATFILE on drive B:. An example of the use of menus is WordStar 3.0 where you enter O for the copy command as shown on a menu, and then are asked "Name Of File To Copy From?" If you enter an acceptable file name, you are asked "Name Of File To Copy To?" With WordStar, if you try to copy to an existing file you receive the message "File FILENAME Exists -- Overwrite (Y/N):. That's error checking, a critical part of user interaction which is a lengthly subject in itself. But how the errors are handled after they are detected should be part of the overall user interface design.

The difficulty of designing the user interface varies with the complexity of the program and the composition of the audience. It is relatively easy to design the interface for a simple program intended for a very small technical audience with similar skills who are expected to use the program frequently enough to remember the commands. For example, a typographer who uses a program to send files to the phototypesetter many times each day should be comfortable with using a few command line parameters because it is a simple one-step operation and they use the same command structure frequently enough to remain familiar with it. Another factor is that they are skilled professionals, working on a terminal most of the day, who have the financial incentive to be fast and accurate, and the use of command line parameters instead of menus will save a significant amount of time for programs which are used many times each day. Menus with beautiful multicolor graphics may impress the novice or the dilettante, but they slow the operation, and in business TIME is MONEY. Therefore, fast is better than beautiful for programs which are used frequently by businesses.

"User/program interaction is a design function at least as important as the problem solving algorithms."

The most difficult user interface to design is the one for a complex program with many steps which is intended for a very broad general market with a wide range of skill levels, where the individual only uses the program occasionally. It is especially difficult when the person (it sure is clumsy trying to avoid using the unisex term 'he', and I refuse to use he/she) also uses other programs with a different command structure so that he becomes confused and can't remember which commands are used for which program.

User Interaction Design vs. Program Design

Most programmers (myself included) concentrate on the "how to accomplish the task" aspect of programming and then tack on the minimum amount of code required to get the input from the user, and there is also usually little or no handling of errors (often just a crash with a warm boot and return to the system prompt).

Like most other people, when I started programming, I concentrated on learning how to write the code to perform the primary functions I needed and I paid little attention to the philosophy of program design and user interaction. Now I can see the need to improve the people interface aspects of my programming, but I haven't located much useful reference material on this subject. I have shelves full of programming books, but they all treat the technical aspects of writing code and technical problem solving details—is this because that was the only type of book which I selected, or is there a lack of material on this subject? It certainly is easier to write about concrete objective technical aspects, than it is to write about the nebulous subjective user interaction interface.

The technical advancements in the next generations of computers with increased speed and memory, plus multitasking and multiusers, will not eliminate the problems of the user interface design—in fact, as more non-computer-experts use these systems, the faults of poor design will become even more apparent and more frustrating. Hot-Shot programmers can write code, and users can complain, but it will take experienced people who have worked with both good and bad programs and who can at least write pseudo code to design the user interface.

A Design Philosophy

Since I can't find the information I need, I'm going to write it! And I'm depending on our readers to argue, disagree, and to offer their experiences and suggestions. Our goal will be to design a range of interfaces for various situations, write system independent pseudo code, and then write the routines in portable high level languages such as Pascal, C, or Modula-2. I think I heard some battered programmers yell "It's impossible to please everyone," and it is impossible to design a reasonable sized interface which will please everyone—but we can at least definewhat should be considered and what methods there are from which to choose.

The first consideration should be the user environment. Is it used for business or pleasure? Is the user experienced or a novice? Is it used frequently or seldom? Is the program simple or complex? Is the terminal local or remote? These and other factors will determine the overall design requirements.

Another very difficult but also very important aspect is satisfying the user's preferences. I feel very strongly that I want to have both command line parameters and menus available without being locked into either one!

Where do We Go From Here?

My plan is to develop a spectrum of user interface and interaction strategies varying from the simple to the complex, plus the criteria for selection. This will be accompanied by pseudo code, code fragments, and useable routines—depending on reader feedback and participation.

This information is of critical importance to everyone who writes a program, and its usefulness will depend on your input. It's up to you to argue with me, disagree with me, expand on what I have to say, or just add your comments.

We start on the next issue as soon as this one is sent to the printer, so write your letters now! ■

A book with

ADVANCED TOPICS in TURBO PASCAL

Turbo Pascal · Advanced Applications

Table of Contents

- Optimization Techniques
- Using the DOS Background Print Spooler
- System Level Tools
- Creating Libraries
- Exploiting Command Line Arguments
- Using a Binary Search Tree
- Techniques for Data Compression
- Claiming CP/M Memory
- Break the 64K Data Limit
- Linked Lists for Data Structuring
- Interrupts from Turbo Pascal
- Calling the DOS Command Processor
- Bit Mapped Graphics
- Teaching an Old Screen New Tricks
- Implementing 2D Core Graphics
- Build a Subset Pascal Compiler

Order ***Turbo Pascal · Advanced Applications*** for $19.95 postpaid in USA; with MS DOS disk, $32.95. Add $3.50 for surface shipment to Canada or other countries; air rates on request. Order from **Rockland Publishing, Inc.**, 190 Sullivan, Suite 103, Columbia Falls, MT. 59912. Visa or Mastercard accepted. Phone orders , call (406) 257-9119. **Free information is available.** Dealer inquiries welcomed.

"...packed with good advanced technical information."

NOT ANOTHER BEGINNER TUTORIAL

Turbo Pascal · Advanced Applications

Patching Turbo Pascal

Removing the Blasted ":=" Requirement

by Clark A. Calkins, C.C. Software

I like to make things easy on myself. Whether I am programming in BASIC, Fortran, Pascal, or whatever, I want it to be as quick and painless as possible. It especially annoys me to have a compiler force me to type in needless characters. In Pascal, the assignment operator ":=" is just a needless burden. I always forget that blasted colon! After all there is no reason why the compiler needs the colon in this case. Fortran doesn't require it. Neither does Basic. So why does Pascal?

Since I do most of my Pascal programming using Turbo Pascal (on a Z-80 machine), and I do like it, I thought I would at least fix this so that an equal sign would suffice for an assignment operator. Now I know what you are thinking, I will get all confused when I use a different compiler. And you're right. Luckily for me I don't use any other compiler on a regular basis.

Figure 1 - Patch Instructions for Turbo Pascal v3.00A and 3.01A (Z-80)

```
Version 3.00A (Z-80)

A>DDT TURBO.COM
DDT VERS 2.00
NEXT   PC
7A00  0100
-S6F7F
6F7F 76 0B
6F80 6E 6F
6F81 82 18
6F82 7F F3
6F83 C8 .
-GO
A>SAVE 121 TURBO.COM

Version 3.01A (Z-80)

A>DDT TURBO.COM
DDT VERS 2.00
NEXT   PC
7900  0100
-S6F61
6F61 58 ED
6F62 6E 6E
6F63 64 18
6F64 75 F3
6F65 C8 .
-GO
A>SAVE 120 TURBO.COM
```

The patches described in Figure 1 will work for Turbo Pascal version 3.00A and 3.01A but not for version 2 or less. Use DDT as shown to make the changes (they really are easy!). Note that the characters that you are to type are shown in **bold** type. After changing the compiler, type in the following sample program just to make sure it runs okay.

Sample Program

```
program Demo;

var
  i:integer;
  Miles,Gallons,mpg:real;

begin
for i=1 to 10 do writeln('testing....',i);
Miles:=287.0;
Gallons=12.3;
mpg=Miles/Gallons;
writeln(' current miles per gallon =',mpg);
end.
```

As you can see, the colon is now optional in assignment statements and also the "for" statement. It was left in one statement just to check that the compiler will still allow it as it did before. We want to be compatible with all of our previously written programs, of course.

One side effect of this change is that the error message displayed when an improper assignment operator is entered is number 06, "'=' equals expected" rather than number 07, "':=' expected". Oh well, I can live with that.

Although it would be more involved, this patch could have been associated with a new compiler option so it could be turned on or off. This would require the compiler to be extended to make room for the added instructions. Maybe next time I'll go into how this can be done. ■

(Continued from page 5)

indicate that really new uses are being made of the newer CPUs. Instead, I feel that lots of programs that have worked in the past are being adapted to the newer applications, and that means using the true and stable Z80 cards.

What I feel is important to keep in mind is that most applications in industrial use, like lab work, need special I/O operations, and the actual CPU is secondary. That is and always will be the STD Bus's strongest feature. Yes, it would be nice to see more 64180s running some of those older Z80 programs, but when the collecting data is more important than fancy menus I rather doubt we will see many changes in the ratio of Z80 to other devices. What we really need is some facts about the STD Bus usage, not from the manufacturers, but from the real users.

Bill Kibler

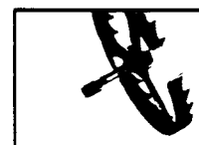
Data Aquisition and Control

I am a recent subscriber and I thought that I would try to tell you a little about myself.

I am a Mining Engineer currently employed in the Coal Industry. I have a strong background in electronics and often make use of this in my work. I subscribe to a number of periodicals and purchase quite a few books. I have been interested in data aquisition and control (discrete and process) for some years. I presently own two PC Compatibles (XT at home and AT at work), an Ampro LB-186 (with SCSI hard disk that I added myself), a Micromint BCC-52 and a number of other things that I put together myself. The common thread in my micros is that they all use Intel MPUs. I can, with difficulty, make prototypes and simple printed circuits. I prefer to purchase bare boards when I can find suitable ones. I am always on the lookout for surplus and bargains. I regard software as a necessary evil.

I will be interested in articles dealing with control applications, the Intel MCS 51/52 family, the 8086 family, the Ampro Little Boards, the SCSI Bus, single board computers, power I/O and similar real time applications.

W.G.



BD Software, Inc., maker of the original

CP/M-80 C Language Development

System, knows

Time is precious

So the compilation, linkage and execution speeds of BDS C are the fastest available, even (especially!) on floppy-based systems. Just ask any user! With 15,000 + packages sold since 1979, there are *lots* of users . . .

New! Ed Ream's RED text editor has been integrated into the package, making BDS C a truly complete, self-contained C development system.

Powerful original features: CDB symbolic source-level debugger, fully customizable library and run-time package (for convenient ROM-ing of code), XMODEM-compatible telecommunications package, and other sample applications.

National C User's Group provides direct access to the wealth of public-domain software written in BDS C, including text editors and formatters, BBS's, assemblers, C compliers, games and much more.

Complete package price: $150.
All soft-sectored disk formats, plus Apple CP/M, available off-the-shelf. Shipping: free, by UPS, within USA for *prepaid* orders. Canada: $5. Other: $25. VISA, MC, COD, rush orders accepted.

BDSoftware, Inc.

BD Software, Inc.
P O Box 2368
Cambridge MA 02238
617 • 576 • 3828

Choosing a Language for Machine Control

Using HTPL as a CNC Language

by Joe Bartel, Hawthorne Technology

At the present time there is a lot of interest in small computer controlled machine tools and other real work applications for microprocessors. Many of these fit in the category of light automation, while others fall into the category of robotics.

A moving platform or other toy like creation may be fun for learning basic concepts but they can't do anything to earn their keep. Microcomputers took off and got big, because as well as being fun to work with, they could do enough useful work to justify the cost. If small robotic or automation projects can be built that can pay their own way, then home automation can grow like home computers did.

If you want to program a computer to do accounting there are many languages that are well adapted to the task, such as data bases or spread sheets. If you want to do graphics there are tools built into many modern dialects of BASIC that make it easy. There are also graphics built into many current home computers. All the things you can do with the standard computer outputs like video or paper are relatively easy.

Control of a machine is not so easy. While most printers are pretty much the same this cannot be said for machine tools. Most machine tools are expensive relative to simple office tools like printers or plotters, and there are not as many of them so it is not as easy to get time to play with one. Many automation projects are flexible but are even less standard than lathes or milling machines.

When contemplating how to program a machine, two approaches come to mind. You can input the desired part using a graphic technique or you can use a programming language. The use of graphics to input a physical shape and then edit it with a mouse is very appealing. This is one of the easiest ways for an operator to enter the data, however this is very hard to program. In small volumes or hobby kinds of machines the effort saved in inputting the part is more than lost in the cost and complexity of the graphics input and editing. Also you need the graphics output and the graphic input.

A non-graphic language means that the programs or parts can be edited with a common text editor, but creating a new language for a special application is very expensive. Also it is hard to know before starting what will be needed in the language or what is economical to provide and maintain. If a flexible existing language can be extended to cover the task, the cost is much less. The language I have chosen to use as an example for control is HTPL. FORTH is a language that is similar and would also be good for this. If parts seem rough it is because I am not a machinist but a systems programmer.

Every command/control language has certain features in common. There must be a way to send messages to the operator and to get information from the operator. There are objects that can be defined and operations that can be performed on the objects. There also must be some way for the language to be entered into the system, edited, saved, and translated into something meaningful. From an implementation standpoint the most important feature is how parameters are passed to subroutines.

In the past BASIC or a language like BASIC has been used as a base from which to start. The advantage is that the form of the program is familiar to most of the users of the language. A big disadvantage is that it is hard to modify a language like BASIC. When looking at Pascal or Modula the task is even larger. Much of the complexity of these involve features that are not needed by the machine tool programmer.

For any given size translator, an RPN language can generate more efficient code and a more reliable language processor. The form of the verbs (operators) in an RPN language mean that those defined by the user look very similar to and act similar to those provided by the language developer. A smaller language also means that there are fewer rules that need to be used in translating the high level code into machine actions. For machine control on an experimental basis this is very important. Each numerical control tool will need to have the language customized for it.

In an RPN language, the action word always finds the values to work with on the evaluation stack. Another advantage is that the user defined operating words look the same as the built in words. The HTPL language differs from FORTH in making more use of variables in memory like BASIC. Also the structure is very similar to more conventional languages like BASIC.

When we created HTPL we decided to make it a compiled language rather then an interpreted or threaded code language. When using a RAM disk a two pass compiler can be as fast as an incremental compiler. The use of a two pass approach also means that procedures can be forward referenced and it can use other language features that are hard or impossible with an incremental compiler.

This example assumes that we will be working with a drill press on an XY table. What we want to do is to program a pattern and then let the machine drill the holes. We will also assume that all the standard HTPL words are available and will only define those that need to be added to create a usable parts programming system. The actual coding for the new action words will not be given because they would probably be done in assembly and would be different for each machine. Also we did not actually build themachine.

A typical action word that moves the drill to a new location will depend on what kind of motor control is used. Also it will depend on how and where its address is.

The sample programs show the use of movement and the use of procedures. A complex part that has a repeating pattern can be made by calling a predefined procedure. If a change is needed to the pattern then by changing it all the other parts change in the same way. In our example a hole pattern is used 4 times but in a different place each time. If this was used for circuit boards then different patterns could be defined for different ICs.

Sample Program

```
orgset    ( -- )          set the origin to 0,0
setrpm    (rpm -- )       set the rpm for the drill
coolon    ( -- )          turn on coolant
cooloff   ( -- )          turn coolant off
position  ( xx yy -- )    move the drill to an absolute location
move      ( xx yy -- )    move the drill to a relative location
getpos    ( -- xx yy )    find out where we are
drill     ( -- )          drill a hole
stop      ( -- )          stop the machine


program   ( main part of program )
  orgset
  6 !count1
  repeat
    holepat
    @count1 -1 dup !count1 =0 until
  0 0 position
  stop
  "-- PART DONE --" message
  end

proc holepat ( drill hole pattern )
  1200 setrpm coolon
  drill        200 0 move drill
  200 minus  0 move drill   200 0 move drill
  cooloff end
```

We also have an example of a loop. The hole pattern is repeated 6 times on the part. If a larger part needed the same pattern then it could as easily produce the pattern any number of times.

The list of control procedures is an example of what could be done. Most of these will have to be created in assembly language to control the actual motors of the drilling machine. If the use of these procedures is standardized then it should be possible to develope a library of programs that can be moved between simple home produced CNC machines.

In a future article we will examine how to make the hardware that the program is controlling. We will also look at what is needed to produce a control program for a lathe or a milling machine or a robot.

Editor's Note: We will be using the HTPL language for both the TinyGiant 68000 SBC and various CNC and control programs. A complete description of HTPL can be obtained from Hawthorne Technology, 8836 S.E. Stark, Portland, OR 97216 phone (503) 254-2005. ■

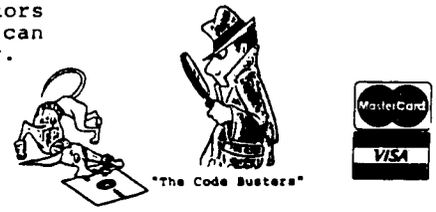
Advertiser's Index

AMPRO Computers..... 26, 27
BD Software..... 40
Bear Electronics..... 23
Bersearch..... 8
BV Engineering..... 36
C User's Group..... 43
C.C. Software..... 42
Computer Journal..... 34, 35
Echelon, Inc..... 2, 21
Hawthorne Technology..... 32
Kenmore Computer Tech..... 23
Micromint..... 5
Rockland Publishing..... 38
Sage Microsystems East..... 18
Silicon Valley Surplus..... 11
Turbo Power..... 33
Ultra Link..... 16

Ever Wondered What Makes *TURBOPASCAL* Tick?

Source Code Generators by C. C. Software can give you the answer.

"The darndest thing I ever did see..."
"... if you're at all interested in what's going on in your system, it's worth it."
Jerry Pournelle, BYTE, Sept '83



The SCG-TP program produces fully commented and labeled source code for your TURBO-Pascal system. To modify, just edit and assemble. Version 3.00A (Z80) is $45. SCG's available for CP/M 2.2 ($45) and CP/M+ ($75). Please include $1.50 postage (in Calif add 6.5%).

C. C. Software, 1907 Alvarado Ave. Dept M
Walnut Creek, CA 94596 (415)939-8153

CP/M is a registered trademark of Digital Research, Inc.
TURBO Pascal is a trademark of Borland International

(Continued from page 44)

with the newly added words and saved you from loading them each time you start the tutor program, but then you would not learn how to load programs. Forth can only be learned by using it, take it from one who tried to only read about it.

Let's discuss a use which Art suggested, a business package to keep mailing labels and update them. The first step in doing any program would be defining the task; the label sizes, how many lines, how many labels, updating problems, and so on. Now that we have the objective defined, we look at how Forth can help; should we use the regualr screen system, or just open files and move records around. We might also be more concerned about using an existing printer and whether it will handle the labels correctly. What we can do first then is to work on the problem areas and leave some of the others till later. Take the printing problem, I would write the words to print labels using the internal editor, and save them in one or two screens. I would test them on a screen of addresses and leave the file words till later. This would allow me to see if I have any problems printing or handling a few labels. I also get to learn if some problems might arise later that I didn't see eariler. (For a good explanation on this bottom up programing style read "THINKING FORTH" by Leo Brodie, it is full of how and why of programming.)

You use this process until the entire program is written, with each new word tested before writing the next one. When finished you have a running program, usually with few bugs (remember you tested each new word!). What you have will be several screens of new words that are added to the regular Forth dictionary to handle the desired task. You would first use routines that worked using regular Forth operations, then replace them with your own routines (like a special line editor), till you got a program that has the Forth aspect hidden to the user. This reminds me that the first wordprocessor for the PC was a Forth program, with the Forth operations hidden from view. If I was doing this for my office, I would just have the Forth system auto load the program, while allowing some password exit for maintenance. As a saleable item, however you would most likely want users kept out and would generate a special version of Forth.

What we have are three ways to create our mailing label system. The first way is to write our extensions in screens (saved on disk as a separate file MAILER.BLK) and then load these extensions to our Forth program (OPEN then 1 LOAD) when we want to run the application. This allows easy access to the words and their definitions when problems arise. It requires that the operator have some knowledge of the system, such as how to load an application. Once in the program, no further Forth understanding is needed. This is basically how my tutor program works.

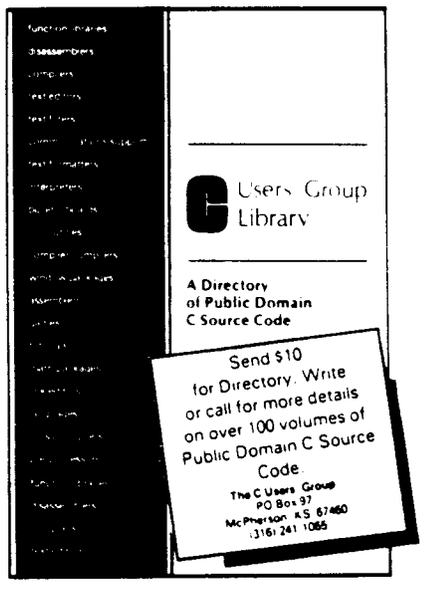
The next way is to change the Forth system sign on screen to auto load the application, much as the trained operator would do. This would still give you the use of Forth if you left some form of hooks to get to it. You would do this by saving the modified Forth system, using Forth's SAVE FORTH word. This saves the new words and the system image, of which you would have modified the booting screen to open your program file and load the extensions. You could also just load ALL the new words and save this image. The difference being, a separate source file (that can be updated) in the earlier version, while the latter requires resaving the entire Forth system if you have changes to make.

The last way has to do with meta compilation, or using your own system to genreate a new system. This is the area of Forth I plan on studying next. In this operation we generate an entire system with only those functions we need to run our application. You start with the kernal and add only the utilities or screens needed by the program. The PC wordprocessor was done in this manner, producing a product hard to tell different from any other application. What is different however is the development time in getting to this phase. To get to this stage, we were able to test small portions using regular Forth. Try our package out as extensions to regular Forth. Do beta testing using a saved Forth system and lastly market a smaller non-Forth looking product.

When it came to writing my tutor article, I guess I didn't cover all the ways to use the program. One reason for this is the many options you have. The other reason is the ease with which you can load and run the package. As so often happens you sometimes can not see the forest for the trees. Forth at times can be so simple you expect to have to do more not less.

Forthing Ahead

I am making plans to resurrect my Forth based monitor program, only this time I am using some pr en hardware. I have a public domain Forth monitor for the Xerox 820, but no source code, it is a bit short, and without the features I want. We also had a reader indicate wanting to see some more Z80 stuff and so I think making a Forth monitor for the 820 will be an ideal project. My problems with the 68K monitor project were hardware, not Forth, but that may get off the ground yet. I have swapped the 6809 system for a better S100 bus system and will be using it for the 68K projects (maybe even a Novix 4000). I have the HTPL K-OS ONE operating system and will port it over soon. I have talked to Hawthorne Technology and we are trying to get more documentation on porting their system for those having troubles. Hopefuly I will have more to say on this later. In fact...until later..enjoy. ■



THE COMPUTER CORNER

A Column by Bill Kibler

Time keeps moving on it seems, even when you would rather it didn't. I just got back from a school in Los Angeles and now have to make up for being away for a week. I had planned on doing this article last week, but a surprise trip to L.A. put a crimp in the plans. The trip however actually added to what I have to say.

The school was on a GE industrial computer, one that I have been maintaining for three years. I knew there was a lot I didn't understand about the system and the school helped in that respect. The school also confirmed my feelings about GE and their equipment—don't buy it if you can avoid it. This statement really has to do with their documentation, and as you know, I am a stickler for good documentation. GE has the worst documents I have ever used, except for those unreadable foregin translations. GE has their own way of documenting their equipment and the school was mostly how to interpret what they did. The rest of the world puts a slash mark through their zeros, not GE they do it to the letter O. To make matters worse, their manuals are assembled collections of all engineering drawings used in every version of the machine. This means you may have to thumb through all 500 drawing until you find the one you are after. Indexes are useless because some of the drawings are numbered and many are not. It is all really a waste of time, when compared with other companies who give you "as built" prints, or properly organized manuals which you can use quite easily. The last blow of the whole affair is that GE charges you to learn how to use their books, when most compaines have free training (you pay your own expenses).

Enough of the industrial computer world, let's see what is happening in the business computer industry. The open MAC is out and the rest of the 68K units will have their 68020 machines out soon. I think the open MAC will help out a lot, but it took far too long for its release, and the price is a bit steep for my pocket book. I am waitng for the MEGA ST, that is the Atari ST in a PC style box with 1, 2, or 4 megs of memory (at half the price of the MAC). As you may remember I feel the 68K series of machines is far superior to all the 8086 based machines. I had a chance to see this when running windows on a PC and compared the same operation on the ST. The ST beats the PC hands down, it is faster and produces better windows and utilities. The machines however that have my interest currently are the boards using the Novix 4000.

My interest in Forth has gotten a boost over the past few months with the release of the Novix chips. These chips run Forth as their only instruction set. I recieved some fact sheets on them and found out that they can also run many instructions combined as one instruction. This means that one machine cycle will perform several operations all together (actually in parallel) as if it were one Forth word. The speed advantages are considerable and can produce around 4MIPS operation at 4 MHZ clock speeds. Why I am interested in this chip has to do with my interest in computers that can be used for teaching. I am trying to decide on what would make the most ideal teaching hardware system, and it is a toss up between the 68020 and a Novix 8000 (32 bit version).

Speaking of teaching, my tutor program got some response from Art, our editor. Let me make it clear that you don't have to be an editor to have questions about our articles. When ever something seems a bit vague, drop us a note and I will explain it. Art had trouble understanding how to create applications using Forth. I think the problem has to do with having gotten used to hard to use systems. Let me explain.

Forth as a Concept

When programs were first written, many of the higher level languages were little better than assemblers as far as what you could or could not do. I remember when Turbo Pascal first came out and what a pleasure it was to move freely between the writing, compiling and error checking stages. Even so, we still had to write an entire program, compile it and then check to see if it worked. The end product usually was a program that either ran under another program (like BASIC) or was COMed to run under the operating system. Even the COMed programs might have embedded overhead, take C, it can have minimum sizes of 32K in some systems. That is 32K even for a one line statement. With all these many steps and problems, most users have gotten used to the rather tedious process of creating programs.

From the start Forth has been different, it was never intended to be just a programming language, but an entire system for solving problems. The idea was to break the problems down into manageable portions, and then provide the tools to solve that problem, all within one system. When I look at Forth from the systems point of view, a lot of the features become most evident. My tutor program tried to show those features and at the same time use them. It does all that but far too easily. Art and many others I am sure have trouble understanding just how to use the tutor, because I did not provide several pages of complex instructions on loading the program.

If you can down load the two programs from the COMPUTER JOURNAL bulletin board, you will find an explanation on loading and operating the tutor program in the file TUTOR.DOC. Those instructions are quite short, just load your F83 (or Forth) program, and at the Forth prompt OK, open the tutor file by using OPEN TUTOR.BLK and when that is done, load screen 1 by entering 1 LOAD. This is all that is needed to get the tutor going, or any Forth application going. Now this is one method and Forth gives you many others as well. What I wish most Forth writers would do is modify their Forth system to include tutorial or help screens. This means that the words become a part of the Forth's regualr dictionary, and when invoked would go to the disk and load the appropriate screen. What you do with my program is extend the dictionary by loading five extra screens of new words. These words then manipulate the text screens to provide the tutor program. I could have also saved the Forth image

(Continued on page 43)